(12)

January 1986 LIDS-TH-1527

A SCHEDULE-BASED APPROACH FOR FLOW-CONTROL IN DATA COMMUNICATION NETWORKS

by

Utpal Mukherji

This report is based on the unaltered thesis by Utpal Mukherji, submitted in partial fulfillment of the requirements for the degree of Doctor of Science at the Massachusetts Institute of Technology, Laboratory for Information and Decision Systems with partial support provided by the Advanced Research Projects Agency under contract ONR/N00014-84-K-0357 and the National Science Foundation under grant NSF-ECS-8310698.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Encl (1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO. AD-A163 253

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A SCHEDULE-BASED APPROACH FOR FLOW-CONTROL IN DATA COMMUNICATION NETWORKS

5. TYPE OF REPORT & PERIOD COVERED

Thesis

6. PERFORMING ORG. REPORT NUMBER

LIDS-TH-1527

7. AUTHOR(s)

Utpal Mukherji

8. CONTRACT OR GRANT NUMBER(s)

DARPA Order No. 3045/2-2-84
Amendment #11
ONR/N00014-84-K-0357

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Massachusetts Institute of Technology
Laboratory for Information and Decision Systems
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

Program Code No. 5T10
ONR Identifying No. 049-383

11. CONTROLLING OFFICE NAME AND ADDRESS

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

12. REPORT DATE

January 1986

13. NUMBER OF PAGES

117

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Office of Naval Research
Information Systems Program
Code 437
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An approach for achieving user-session packet throughput guarantees and packet intranetwork delay limits is described. Both objectives are important for packetized voice calls. The approach permits flexible usage of link capacity by sessions, which is important for data sessions. Throughput is guaranteed to a session at links in its path by scheduling priority-slots in link-frames for transmission. An extenstion of end-to-end windowing limits the intra-network delay for a session to the sum of, first, the product of the frame-time and the session's window-size, and, second the session's priority-slot schedule-delay.

Go next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (Continued)

An NP-completeness result is proven, showing, for general networks, that the scheduling of priority-slots to obtain minimum sum of schedule-delays is algorithmically hard. Minimun-delay scheduling algorithms for special network classes, and a scheduling heuristic for general networks, are presented. For Poisson packet generation at session rates less than throughput guarantees, limited simulations suggest that low mean values of packet end-to-end delays, relatively insensitive to choice of window-sizes, are obtained even at small but non-zero window-sizes.

A SCHEDULE-BASED APPROACH FOR FLOW-CONTROL IN DATA COMMUNICATION NETWORKS

by

UTPAL MUKHERJI

B.Tech., Indian Institute of Technology, Bombay
(1980)

S.M., Massachusetts Institute of Technology
(1982)

E.E., Massachusetts Institute of Technology
(1984)

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February, 1986

Signature of Author _____
Department of Electrical Engineering and Computer Science
January 6, 1986

Certified by _____
Professor Robert G. Gallager
Thesis Supervisor

Accepted by _____
Professor Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

A Schedule-Based Approach for Flow-Control in Data Communication Networks

by

Utpal Mukherji

Submitted to the Department of
Electrical Engineering and Computer Science
on January 6, 1986
in partial fulfilment of the requirements
for the Degree of Doctor of Science

Abstract

An approach for achieving user-session packet throughput guarantees and packet intra-network delay limits is described. Both objectives are important for packetized voice calls. The approach permits flexible usage of link capacity by sessions, which is important for data sessions. Throughput is guaranteed to a session at links in its path by scheduling priority-slots in link-frames for transmission. An extension of end-to-end windowing limits the intra-network delay for a session to the sum of, first, the product of the frame-time and the session's window-size, and, second, the session's priority-slot schedule-delay. An NP-completeness result is proven, showing, for general networks, that the scheduling of priority-slots to obtain minimum sum of schedule-delays is algorithmically hard. Minimum-delay scheduling algorithms for special network classes, and a scheduling heuristic for general networks, are presented. For Poisson packet generation at session rates less than throughput guarantees, limited simulations suggest that low mean values of packet end-to-end delays, relatively insensitive to choice of window-sizes, are obtained even at small but non-zero window-sizes.

Thesis Supervisor: Dr. Robert G. Gallager
Title: Professor of Electrical Engineering

Acknowledgements

I wish to thank Professor Robert Gallager, my thesis supervisor, for having so kindly provided to me encouragement, guidance, suggestions, ideas, and support, during the course of my research for this thesis. I am most grateful for having had this opportunity of learning from him, for the keen interest he has taken in my progress as a student and in my interests for the future, and for much time and great patience that he has always had for me.

I would like to thank Professor Dimitri Bertsekas, Professor John Tsitsiklis, and Professor Alexander Rinnooy Kan, my thesis readers, for having given of their time and knowledge for my benefit. The many questions put to me by Professor Bertsekas, especially those concerning Chapters 4 and 1, have been most helpful to me. Professor Tsitsiklis I would also like to thank for many valuable discussions and suggestions.

I would also like to thank Professor Michael Sipser for his assistance in sharpening my ideas for Section 3.2 of this thesis report.

I thank Ellen Hahne, fellow-student and office-mate, for having graciously shared with me ideas from her thesis research on flow-control. My many discussions with her have had their influence on this thesis, and her review of the Introduction has been especially helpful.

I have also benefited from consultations concerning this thesis with Professor James Orlin, Professor Pierre Humblet, Dr. Moshe Sidi, Professor Thomas Magnanti, and Professor Christos Papadimitriou. I thank them all. Professor Humblet I would also like to thank for being readily available for consultation on many other occasions during my studies.

This research was conducted at the Laboratory for Information and Decision Systems, and I gratefully acknowledge the support provided by the Defence Advanced Research Projects Agency under Contract ONR/N00014-84-K-0357 and the National Science Foundation under Grant NSF-ECS-8310698.

I wish to thank Professor Sanjoy Mitter, Director of the Laboratory, for much encouragement and keen interest in my progress in research. I would like to thank Professor Alvin Drake, my faculty counselor, for his understanding and able guidance, especially during the early stages of my graduate program. I would also like to thank Professor George Verghese, my master's thesis supervisor, for his encouragement and interest throughout my studies.

I take this opportunity to thank Carey Bunks, Jerry Prince, and System Manager Bob Bruen, for their assistance in setting me up on the Laboratory's new computer system. I thank Arthur Giordani for his expert drawing of the figures for this report. I also thank Jean Regnier, John Spinelli, and Isidro Castineyra, for consultations during the text-editing process for this report on the new type-setting system, and Nancy Young for friendly help in arranging many meetings.

It has been a pleasure to keep the company of my fellow-students in the Laboratory including, among others, Tally Altes, Erdal Arikan, Carey Bunks, Isidro Castineyra, Julio Escobar, Eli Gafni, Ellen Hahne, Dan Helman, Patrick Hosein, Joe Hui, Atul Khanna, Jim Krause, Jay Kuo, Whay Lee, Christophe Pagezy, Abhay Parekh, Jerry Prince, Jean Regnier, Jim Roskind, Clint Roth, John Spinelli, Darius Thabit, Ed Tiedemann, Kevin Tsai, Paul Tseng, Paul Wiley, and Albert Wong. I have especially enjoyed many discussions with Ellen Hahne, Erdal Arikan, Jean Regnier, Isidro Castineyra, Jim Krause, and Kevin Tsai.

I also wish to thank, among others in my Department or Laboratory, Professor John Kassakian, Professor Alan Oppenheim, Professor Robert Kennedy, Professor Martin Schlecht, Dr. Michael Slepian, Dr. Ronald Williams, and Evangelos Milios, for their assistance and interest.

Finally, many thanks are due to many other students and members of the M.I.T. community, for helping make my stay here an enjoyable one.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Users of a data communication network set up sessions to communicate over the network. The messages generated by a user are split into packets at the source, and then carried through the network to the user's destination by the user's session. Packets belonging to a session share, with those belonging to other sessions, the transmission capacities of the links and the storage and switching capacities of the nodes. There is usually much short-term uncertainty in the packet generation rates for the sessions; this can overload the capacities of the network and hence cause instability in the network. The research reported here is aimed at preserving well-defined flows of session packets, while permitting flexible usage of link transmission capacities by the sessions.

The emphasis in most of the work reported in the flow-control literature, as surveyed by Gerla and Kleinrock [13] for example, is on the control of storage congestion. However, this research does not emphasize this problem since storage congestion is becoming less of a problem with the rapidly decreasing cost of providing more storage. Indeed, packet throughputs and delays, for sessions in networks that use flow-control schemes that emphasize storage congestion, are difficult to predict, especially so in the short-term. This research also assumes that the switching capacities of the nodes are large enough to enable full utilization of link transmission capacities. This is an appropriate assumption since processing costs have been decreasing faster than link communication costs.

There are many types of sessions, such as packetized voice, that have relatively stringent and well-defined packet throughput or delay requirements. There is a need for schemes that support sessions for such users. The schedule-based approach defined in this thesis is proposed as a possible means of meeting this need.

1.2 Background

Some flow-control problems taken from the literature are discussed in this section in relation to this thesis. The discussion is set first in the context of conventional data sessions, then in the context of packetized voice sessions, and finally in a more abstract context.

Peak packet generation rates for interactive data sessions greatly exceed their average rates. If link capacity sufficient to accommodate sessions' peak rates were to be dedicated to sessions, the number of sessions that could be accommodated would be severely limited by available link capacity. On the other hand, if link capacity just sufficient to accommodate sessions' average rates were to be dedicated to sessions, large amounts of packet buffering and delay would be required for each session to average the peak levels of its activity over its inactive periods. Occasionally, packets for a session would be greatly delayed while capacity dedicated to other channels lay idle. Flexible usage of link capacity is desirable, particularly on these occasions. Hence, as with most data network flow-control schemes, the schedule-based approach does not dedicate link capacity to data sessions.

Nodal storage limits in data networks can sometimes constrain the usage of link capacity by sessions. A session may monopolize the storage capacity at its source node, locking out packets for sessions that are in transit at that node. Lockouts of sessions at several nodes in a cycle can result in deadlocks where no packets are successfully forwarded by any session. However, such deadlocks can be resolved by reservation of some buffers for appropriately defined packet classes (Raubold and Haenle [28]). Deadlocks should become rare with decreasing storage costs, and with the use of window flow-control schemes. Window flow-control is used in the schedule-based approach, and is outlined below.

A path for routing packets in first-come first-served order from source to destination is assigned to each session, and the number of packets outstanding for the session along its path is limited as follows. In end-to-end window flow-control, a session can have at most a number of packets equal to its window-size, for which its source node has not received packet reception acknowledgements from its destination node (Cerf and Kahn [5], Ahuja [1]). In node-by-node

window flow-control, the same constraint applies between consecutive nodes in a session's path (Rinde and Caisse [30]). In order that nodal storage limits are not exceeded, window-sizes for the sessions should be chosen so that each node can allocate, to each session using it, a number of packet buffers equal to the session's window-size. However, proper choice of window-sizes is difficult because the packet throughput and mean packet delay for each session depends in a complicated manner on the set of all window-sizes chosen (Reiser [29]). The schedule-based scheme makes the problem of choosing window-sizes a less critical one by providing a throughput guarantee, which is not subject to window constraints, to each session. A small session window-size results in small session packet buffering and delay at intermediate nodes in the session's path without any change in the session's throughput guarantee.

The burstiness present in conventional data sessions is also present in voice calls to a lesser degree, since the calls alternate between talk-spurts and silences. As an example, the Time Assigned Speech Interpolation (TASI) system supports a larger number of voice calls on a single link than would have been possible if voice calls had no silences, by inserting talk-spurts for calls in the silence periods of other calls (Bullington and Fraser [4]).

The burstiness of voice calls has also prompted experiments in voice communication over data networks (Weinstein and Forgie [32]). During talk-spurts, voice calls generate packets at regular intervals that depend on the bit-rate used for voice coding, and have corresponding packet throughput requirements (Gold [14]). Packet transit delays from source nodes to destination nodes are also required to satisfy limits that are set so that, at the destinations, packets can be played back at regular intervals. However, large voice packet buffering and delay can result at intermediate nodes in a call's path as link loads approach link capacities. Embedded coding of voice may be used, so that excess delay can be relieved by discarding packets that contain less significant portions of the coded voice signal, at both source and intermediate nodes (Bially, Gold, and Seneff [2]). The schedule-based approach limits packet transit delays from source nodes to destination nodes, and may be a simple approach to limiting voice packet delays. Further, at the possible cost of some additional pre-transmission buffering and delay, a large number of voice calls between the same source and destination can be multiplexed onto

a single schedule-based session that has an appropriately sized throughput guarantee, as in TASI (Weinstein and Hofstetter [33]).

Link-frames have been used in time-division multiplexed circuit-switched voice networks, where a link-frame consists of time-slots that are assigned to the voice calls using the link, and the frame-time duration is the same on all links. Similar link-frames have also been proposed for networks that carry circuit-switched voice and packet-switched data, where the boundary defining the set of voice slots in a frame is moved in order to allow data packets to use voice slots that have not been assigned to voice calls (Coviello and Vena [7]). Data throughput would be increased and data delay decreased if the frequent silences within voice calls were detected, as in TASI, and the silences were used to transmit data packets (Bially, McLaughlin and Weinstein [3], Williams and Leon-Garcia [34]). This is possible in the schedule-based approach, where both data and voice are assumed to be packetized.

The possibility of integrating conventional data sessions and packetized voice calls in data networks has motivated flow-control studies in a more general context. Several flow-control algorithms have been devised that achieve fair allocation of network link capacity to sessions, in the sense of maximizing the minimum session allocation (Jaffe [21], Hayden [18], Gafni [9], Mosely [24], Gafni and Bertsekas [10]). These algorithms are effective under conditions where sessions are set up and disconnected infrequently. The ability of these algorithms to track changes in session activity throughout a network, maximize the minimum active-session allocation, and implement these allocations, may be limited by excessive time-requirements for measurement and communication relating to the algorithms (Oshinsky [26], Mosely [24]). Round-robin scheduling of sessions for packet transmission at links in the network, in conjunction with conventional window flow-control, has been proposed as a simple real-time approach for achieving this objective (Hahne [17]). The schedule-based approach does not necessarily maximize the minimum session allocation, but is a Similar real-time approach that achieves fairness in the sense of providing throughput guarantees to the sessions.

In any flow-control approach, the mean packet end-to-end delay for a session, from generation at source to reception at destination, grows with increasing levels of transmission capacity utilization on the links in the session's path. Proper routing of sessions at set-up time, and control of session set-up attempts, are required for maintaining balanced and limited link utilizations (Schwartz and Stern [31]). The integrated routing and control of session set-up for data, voice, and data and voice together, has been the subject of some research (Golestaani [15], Ibe [20], Gafni [9]). The schedule-based flow-control approach should be used in conjunction with an appropriate routing and control algorithm.

A scheduling problem relating to the packet transit delay limits in the schedule-based approach is considered in this thesis. This work falls in the area of deterministic, time-periodic scheduling. Some work has been reported in this area (Orlin [25]), for example in the context of task scheduling in pipelined computers (Kogge [23], pp. 71-112) and of production scheduling in manufacturing flow-shops (Hitz [19], Graves et al. [16]).

1.3 Outline

Some physical and architectural assumptions are inherent in the schedule-based approach. First, each link can reliably transmit a known amount of information in a link-frame. Frames occur periodically on each link at intervals of one frame-time, where the frame-time is the same on all links. Second, each session has a path set up in the network at session set-up time, along which packets for the session can be routed. Packets for each session are transmitted in the same order on all links in the session's path. Third, information can be communicated to a session's source from its destination for flow-control purposes.

The schedule-based approach is described in Chapter 2 of this report. It is shown that the scheme provides each session in the network with a packet throughput value up to which throughput is guaranteed, as well as an achievable upper bound on the intra-network delay that a packet incurs in the network after starting transmission on the first link in its path. The throughput guarantee is obtained by assigning transmission priority to the session, on each link in its path, in certain time-slots in the link-frames. The upper bound on packet

intra-network delay is obtained by, in addition, limiting the total number of packets for the session that can be in transit in the network at any given time, using an extended version of end-to-end windowing. The delay bound is shown to be a sum of, first, the product of the session's window-size and the frame-time, and, second, the session's schedule-delay. The schedule-delay is a function of the positions of the session's priority-slots in the link-frames.

The scheduling of priority-slots, so as to result in low values of schedule-delays, is considered from an algorithmic point of view in Chapter 3. This problem is shown to be hard for general networks, as is the case for general instances of many other scheduling problems, by proving an NP-completeness result. (Garey and Johnson [12] provide a guide to the theory of NP-completeness.) However, simple and efficient algorithms that result in minimal schedule-delays are presented for special classes of networks. A scheduling heuristic, that can be applied to general networks, is also presented.

Mean packet delays observed in simulations of some networks, with session packets generated in Poisson processes at known rates that are less than the throughput-guarantees, are presented in Chapter 4. The simulation results suggest that low mean values of packet end-to-end delays, from generation at source to reception at destination, are obtained in the schedule-based scheme even with small, but non-zero, window-sizes. Lower values of schedule-delays result in lower mean values of packet intra-network delays, but do not necessarily change mean packet end-to-end delays. In contrast with the low sensitivities of mean end-to-end delays to changes in window-sizes for the schedule-based scheme, these sensitivities for conventional end-to-end windowing, with first-come first-served transmission of packets at links, are observed to be relatively large.

This thesis report concludes, in Chapter 5, with additional comments on the schedule-based approach.

Chapter 2

The Schedule-Based Scheme : Packet-Throughput Guarantees with Upper-Bounded Packet Intra-Network Delays

2.1 Introduction

Throughput guarantees for sessions are realized in the schedule-based scheme by defining a network-wide frame-time, and, at each link in each session's path, assigning packet transmission priority to the session in a time-periodic manner, once per frame-time. An extended version of end-to-end windowing maintains the throughput guarantee for each session, while upper-bounding the intra-network packet buffering for the session. The throughput guarantee and bounded buffering, together, result in an upper bound on the packet intra-network delay for the session. These operations are explained in this chapter, assuming the following network model.

a) All packets, for all sessions, are the same number of bits in length. Each link is time-slotted, with slot duration equal to the packet transmission time. Packet transmissions begin at the start-times of slots. Links are error-free, and can reliably communicate packets, one per time-slot. All links are equal in bit-speed, so that the slot duration on each link is the same. This duration is one time-unit.

b) All packets belonging to a session are routed along the same path in the network, where the path is chosen at session set-up time. The packets belonging to the session are transmitted in the same order on all links in the session's path. Packet propagation delays at all links, and packet switching delays at all nodes, are zero.

c) Window-tokens are returned to a session's source node from its destination node, for all sessions.

2.2 Realization of throughput guarantees

The frame-time for the network is chosen equal to T time-units, and a link-frame of length T slots is defined on each link. Each session is guaranteed a throughput of 1 packet per T time-units, as follows. At set-up time, a session's path is chosen subject to the restriction that the number of sessions sharing any link is at most T, and the session is assigned priority of transmission in one slot in the frame on each link in its path. Link-frames are repeated on their respective links at intervals of one frame-time. Thus, each session is guaranteed packet transmission priority, on each link in its path, at intervals of T time-units, making feasible a session throughput of 1 packet per T time-units.

2.3 Upper-bounding of intra-network packet-buffering

Consider a session s that shares a link l with $T-1$ other sessions. If each session generates packets in a random process, with mean packet-generation rate less than but approaching $1/T$ packets per time-unit, the packet-buffering requirements for session s typically approach infinity. The storage is required along the part of the session's path that is up-stream from the link l. If the link is not the first in the session's path, this storage may be required at an intermediate node in the path. The storage requirements for the session can be reduced at the intermediate nodes in its path, and concentrated mainly at its source node, by using the following extended version of end-to-end windowing, which upper-bounds the packet-buffering for a session at all its intermediate nodes combined.

Real tokens for the session, equal in number to the session's window-size, are created at session set-up time, at the session's source node. As in conventional end-to-end windowing, these tokens are returned to the session's source node after they reach its destination node. If a packet belonging to the session is to be transmitted on the first link in the session's path in a non-priority slot, in which the session does not have pre-assigned transmission priority, the packet is required to acquire a real token at the source node, and deposit the token at the opposite node for the link upon reception there. While the session's usage of non-priority slots on the first link in its path may be blocked temporarily for a lack of real tokens, its usage of

its priority-slots is not blocked on any link in its path, so that its throughput guarantee of one packet per frame can be maintained. If a packet belonging to the session is transmitted on the first link in the session's path in a priority-slot, it is assumed to carry a fictitious token for the session over the link, and deposit the token at the opposite node for the link. Unlike real tokens, fictitious tokens are removed from the network when they reach the session's destination node.

Since each packet belonging to the session carries a token for the session over the first link in the session's path, the number of such packets that are received at the first intermediate node in the path equals the number of times that tokens for the session, of real or fictitious type, are received at that node. When a packet belonging to the session is transmitted on a subsequent link in the path, it is again required to carry a token for the session, real or fictitious, over the link and deposit the token at the opposite node for the link. Thus, the number of the session's packets at an intermediate node in the path, equals the number of real and fictitious tokens for the session at the node.

The number of real tokens present for the session, at all intermediate nodes combined, is at most equal to the window-size for the session. The number of fictitious tokens is bounded as follows. The maximum rate at which fictitious tokens are carried over the first link in the session's path equals one per frame. The session does not have any guarantee that its packets can be transmitted in non-priority slots on any link in its path. A fictitious token is selected if possible, rather than a real token, when a packet is transmitted in a priority-slot on a subsequent link. This ensures that the number of fictitious tokens at intermediate nodes remains upper-bounded, even in the extreme case for which the fictitious token rate on the first link is one per frame and only priority-slots, once per frame, are available on subsequent links. In the extreme case above, this rule forces the real token circulation rate to zero, and the fictitious token rate on all links in the path equals the guaranteed packet rate of one per frame. While real and fictitious tokens are thus sometimes carried in different order on different links in the session's path, packets for the session are always transmitted in the same order on all links in the path. The token-usage algorithm for a session is summarized in Figure 2.1.

Type of token as function of link and slot

	Priority slot	Non-priority slot
First link	Fictitious	Real
Subsequent link	Fictitious, if any; otherwise, Real	Fictitious or Real

Number of Real tokens equals Window-size

Real tokens are returned to source from destination

Fictitious tokens are always present at source

Figure 2.1 The Token Usage Algorithm for a Session

2.4 Upper bound for packet intra-network delay

The intra-network delay, for a packet belonging to a session, is the time from the start of packet transmission on the first link in the session's path, to the finish of packet reception on the last link in the path. The packet intra-network delay is equal to 1 if the number H of links in the path is 1. Otherwise, as shown here, the delay is upper-bounded by the sum of, first, the product wT of the session's window-size w and the frame-time T, and, second, the session's schedule-delay.

The schedule-sequence, schedule-delay, and schedule-wait, for a session, are defined as follows. Assume, for purposes of definition, that, first, the session can transmit only in priority-slots on each link in its path, second, the session has only one packet to carry through the network, and, third, the packet is transmitted on the first link in the priority-slot that starts at time σ_1, $0 \leq \sigma_1 < T$. Let σ_h, $2 \leq h \leq H$, denote the start-time of the priority-slot, on the h-th link in the path, in which the packet is transmitted on that link. Since priority-slots

recur on each link at intervals of the frame-time T, the packet waits less than T time-units at each intermediate node in the session's path, i.e., $\sigma_h + 1 \leq \sigma_{h+1} < \sigma_h + 1 + T$, $1 \leq h \leq H-1$. The schedule-sequence for the session is the sequence $\sigma_1, \sigma_2, \ldots, \sigma_H$. The schedule-delay for the session is $\sigma_H - \sigma_1 + 1$, the intra-network delay for the packet. The schedule-wait for the session is $\sigma_H - \sigma_1 + 1 - H$, the difference between the schedule-delay and the H time-units used for transmission of the packet. Since there are $H-1$ intermediate nodes in the session's path, the schedule-wait is less than $(H-1)T$. Hence, the schedule-delay is less than $(H-1)T + H$, and the delay-bound to be shown, $(\sigma_H - \sigma_1 + 1) + wT$, is less than $(w + H - 1)T + H$.

A heuristic argument is presented below, for a session with $H \geq 2$ links in its path, to show that the intra-network delay for any packet belonging to the session is upper-bounded by $(\sigma_H - \sigma_1 + 1) + wT$. The proof is presented in Appendix A.

Worst-case scenario when $w = 0$ (a heuristic argument) :

Since the session has no real tokens, all packets use priority-slots when they are transmitted on the first link in the path. The worst-case intra-network delay for the session arises if the maximum possible number of packets for the session are transmitted on the first link, and slots are available to the session on subsequent links at the minimum possible rate. Therefore, assume that the i-th packet, $i \geq 1$, for the session starts transmission on the first link at time $\sigma_1 + (i-1)T$, and that only priority-slots are available to the session on subsequent links.

In this scenario, the first packet for the session starts transmission on the second link in the path at time σ_2. Also, it follows that the i-th packet, $i \geq 2$, starts transmission on the second link at time $\sigma_2 + (i-1)T$. In general, the i-th packet, $i \geq 1$, starts transmission on the h-th link, $1 \leq h \leq H$, at time $\sigma_h + (i-1)T$. The intra-network delay for the i-th packet, $i \geq 1$, is $[\sigma_H + (i-1)T] - [\sigma_1 + (i-1)T] + 1$, or $\sigma_H - \sigma_1 + 1$, which is the delay-bound to be shown for $w = 0$.

Worst-case scenario when $w > 0$ (a heuristic argument) :

If the maximum possible number of packets for the session are transmitted on the first link, subject to the condition that only priority-slots are available to the session on subsequent links, then, first, the fictitious tokens for the session start transmission on the links in the path at the same time-instants as the packets, or fictitious tokens, do in the $w = 0$ worst-case scenario, and, second, the w real tokens for the session are backlogged at the second link in the path. Thus, this scenario results in the maximum numbers, and maximal delaying, of both fictitious and real tokens, and hence packets, in the network for the session.

In this scenario, packets for the session can use only priority-slots when they are transmitted on the first link in the path. A packet that starts transmission on the first link at time $\sigma_1 + (i-1)T$ cannot start transmission on the second link at time $\sigma_2 + (i-1)T$, but must do so after the w packets belonging to its session, that are enqueued in front of it at the second link, are transmitted on the second link. Thus, the packet starts transmission on the second link at time $\sigma_2 + (i-1+w)T$, carrying a fictitious token, and starts transmission on the h-th link, $h \geq 3$, at time $\sigma_h + (i-1+w)T$, carrying the same fictitious token. The intra-network delay for the packet is $[\sigma_H + (i-1+w)T] - [\sigma_1 + (i-1)T] + 1$, or $(\sigma_H - \sigma_1 + 1) + wT$, which is the delay-bound to be shown.

Chapter 3

Scheduling of Priority-Slots : An NP-Completeness Result and Some Algorithms

3.1 Introduction

The schedule-based scheme, as described in Chapter 2, upper-bounds the packet intra-network delay for each session by the sum of, first, the session's schedule-delay, and, second, the product of the session's window-size and the frame-time. The schedule-delay for a session is determined by its schedule-sequence, i.e., by the position of its priority-slot in the frame on each link in its path. The scheduling of priority-slots, so as to result in low values of schedule-delays, is considered from an algorithmic point of view in this chapter.

A schedule that has non-integer valued slot start-times can be efficiently transformed into a schedule with only integer-valued start-times, without increasing the sum of schedule-delays; the procedure is described in Appendix B. Accordingly, attention is restricted to the construction of schedules that have integer-valued slot start-times.

First, for networks that have frame-time equal to 3 time-units, the problem of deciding whether schedules that have all schedule-waits equal to zero exist, is shown to be NP-complete. In the theory of computational complexity, it is conjectured that there do not exist any polynomial-time algorithms for solving all instances of a problem that is NP-complete (Garey and Johnson [12]). Hence, some special classes of networks are considered next, for which linear-time algorithms are presented for computing schedules that have the minimum sum of schedule-waits. Finally, a scheduling heuristic that can be applied to general networks is presented together with upper bounds on the resulting sum of schedule-waits.

3.2 An NP-completeness result

Let S denote the set of sessions, and L the set of links, in a network that has frame-time equal to T time-units. The paths for the sessions are known, and the number of sessions

sharing any link is at most T. Let H^s, $1 \leq s \leq |S|$, denote the number of links in the path for the s-th session. The schedule for the network is the set of schedule-sequences for the sessions in S. More precisely, an integer schedule σ is a mapping σ_h^s, $1 \leq s \leq |S|$, $1 \leq h \leq H^s$, into the set of integers, that satisfies the following conditions 3.1.

$$0 \leq \sigma_1^s \leq T-1,\ 1 \leq s \leq |S|. \tag{3.1a}$$

$$1 \leq \sigma_h^s - \sigma_{h-1}^s \leq T,\ 1 \leq s \leq |S|,\ 2 \leq h \leq H^s. \tag{3.1b}$$

If the l-th link is the h-th sequential link in the s-th session's path and also the h'-th sequential link in the s'-th session's path, $1 \leq l \leq |L|$, $s \neq s'$, $1 \leq s, s' \leq |S|$, $1 \leq h \leq H^s$, $1 \leq h' \leq H^{s'}$, then

$$\sigma_h^s \neq \sigma_{h'}^{s'}\ (modulo\ T). \tag{3.1c}$$

The sum of schedule-waits for a schedule σ is

$$W_\sigma = \sum_{s=1}^{|S|} (\sigma_{H^s}^s - \sigma_1^s + 1 - H^s). \tag{3.2}$$

The following 'Network 3-Periodic Zero-Wait Scheduling' decision problem is shown here to be NP-complete : for a network instance I that has frame-time $T = 3$ does there exist an integer schedule σ that has sum of schedule-waits $W_\sigma = 0$? Some corollaries of this NP-completeness result are presented in Appendix C.

Theorem 3.1:

The 'Network 3-Periodic Zero-Wait Scheduling' problem is NP-complete.

Proof: The theorem follows from a) and b) below.

a) The 'Network 3-Periodic Zero-Wait Scheduling' problem is in the class of problems NP. This is shown as follows.

Assume that there exists a zero-wait integer schedule σ for the network instance I. Then, the validity of conditions 3.1 for schedule σ, and of the value W_σ given by eqn. 3.2, can be checked in polynomial-time. Therefore, the problem is in the class NP.

b) The NP-complete 'Graph 3-Colourability' problem is reducible to the 'Network 3-Periodic Zero-Wait Scheduling' problem, in polynomial-time. This is shown below.

Let V denote the set of vertices, and E the set of edges, of a graph. A 3-colouring f for the graph is a mapping f_v, $1 \leq v \leq |V|$, into the set $\{0,1,2\}$, of 'vertex colours', that satisfies the following condition. If the e-th edge is $[u,v]$, i.e., incident on the u-th and v-th vertices, $1 \leq e \leq |E|$, $u \neq v$, $1 \leq u,v \leq |V|$, then

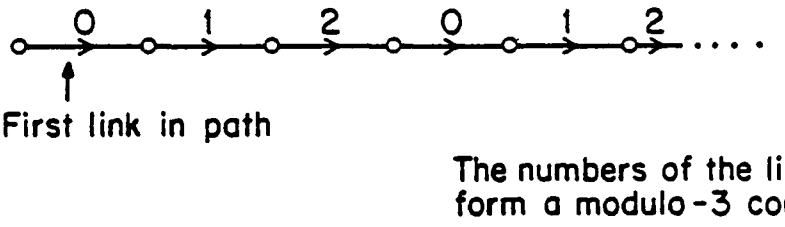
$$f_u \neq f_v. \tag{3.3}$$

Consider the 'Graph 3-Colourability' problem : for a graph instance $G = (V,E)$, does there exist a 3-colouring f ?

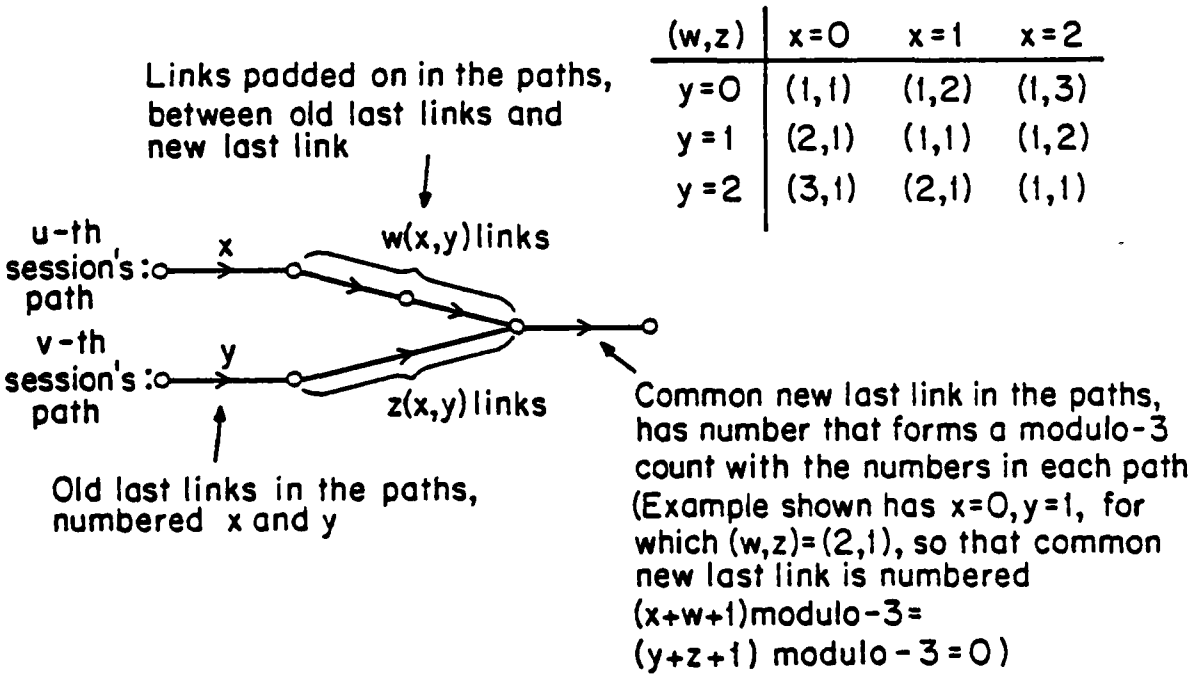
The procedure described next is used to construct a network instance I corresponding to the graph instance G, with the following properties. The network has frame-time $T = 3$. There is a one-to-one correspondence of sessions in the network to vertices in the graph. Each link in the network lies in one or two of the paths for the sessions. All links are numbered 0, 1, or 2. As shown in Fig. 3.1a, the numbers of the sequential links in each session's path constitute a modulo-3 count, with the first link numbered 0. The construction procedure is as follows.

i) For $1 \leq v \leq |V|$, repeat the following. Construct a separate first link, numbered 0, for the v-th session's path.

ii) For $1 \leq e \leq |E|$, repeat the following. If, in graph G, the e-th edge is incident on the u-th and v-th vertices, then extend the u-th and v-th sessions' paths so as to meet in a common link, as shown in Fig. 3.1b. If the old last links in the u-th and v-th sessions' paths are numbered x and y respectively, then $w(x,y)$ and $z(x,y)$ links are padded on in the



(a) The numbers of the links in a session's path



(w,z)	x=0	x=1	x=2
y=0	(1,1)	(1,2)	(1,3)
y=1	(2,1)	(1,1)	(1,2)
y=2	(3,1)	(2,1)	(1,1)

(b) The extension of the u-th and v-th sessions' paths corresponding to edge $[u,v]$

Figure 3.1 The Structure of the Network I in the Proof of Theorem 3.1

respective paths, so that the number resulting for the new common last link forms a modulo-3 count with the link-numbers in each path.

This is a linear-time construction procedure; the constructed network I has $|V|$ sessions and at most $|V| + 5|E|$ links. As shown next, a zero-wait integer schedule exists for network I if, and only if, a 3-colouring exists for graph G. Thus, the 'Graph 3-Colourability' problem is reduced to the 'Network 3-Periodic Zero-Wait Scheduling' problem, in polynomial-time.

Suppose f is a 3-colouring for G. Then, as shown below, there exists a zero-wait integer schedule for I. Define σ by $\sigma_h^v = f_v + h - 1,\ 1 \leq v \leq |V|,\ 1 \leq h \leq H^v$. Then, the integers σ_h^v satisfy conditions 3.1a and 3.1b, and the value W_σ given by eqn. 3.2 is 0. Condition 3.1c is verified as follows. If a link is the h_u-th and h_v-th sequential link in the u-th and v-th sessions' paths respectively, then it is the common new last link resulting from the extension corresponding to the edge $[u, v]$ in G. Hence, $f_u \neq f_v (modulo\ 3)$, and $h_u = h_v (modulo\ 3)$. Therefore, $\sigma_{h_u}^u \neq \sigma_{h_v}^v (modulo\ 3)$, and condition 3.1c is satisfied. Thus, σ is a zero-wait integer schedule for I.

Conversely, suppose that σ is a zero-wait integer schedule for I. Then, as shown below, there exists a 3-colouring for G. Define f by $f_v = \sigma_1^v,\ 1 \leq v \leq |V|$. Then, each number f_v is 0, 1, or 2. Condition 3.3 is verified as follows. If $[u, v]$ is an edge in G, then there is a link in I that is the new last link resulting from the extension corresponding to edge $[u, v]$. Let this link be the h_u-th and h_v-th sequential link in the u-th and v-th sessions' paths respectively. Then, $\sigma_{h_u}^u \neq \sigma_{h_v}^v (modulo\ 3)$, and $h_u = h_v (modulo\ 3)$. Since $W_\sigma = 0$, $\sigma_{h_u}^u = \sigma_1^u + h_u - 1$ and $\sigma_{h_v}^v = \sigma_1^v + h_v - 1$. Therefore, $\sigma_1^u \neq \sigma_1^v (modulo\ 3)$, and condition 3.3 is satisfied. Thus, f is a 3-colouring for G.

This completes the proof of Theorem 3.1.

3.3 Some scheduling algorithms

The discussion of scheduling algorithms is facilitated by the following definition of the link-precedence graph for a network. There is a one-to-one correspondence between links in

the network and vertices in the link-precedence graph. There is a directed arc $a = (i, j)$ from the i-th vertex to the j-th vertex in the link-precedence graph if, and only if, at least one session in the network has the corresponding links, in the order (i, j), as consecutive links in its path. The weight $w_a = w_{i,j}$, of arc $a = (i, j)$ of the link-precedence graph, is the number of sessions that have the corresponding links as consecutive links in their paths.

If the link-precedence graph is a tree, then the following algorithm A_{tree} constructs an integer schedule that has sum of schedule-waits equal to 0, in linear-time. A_{tree} constructs the schedule link by link, each time scheduling all priority-slots that are to be scheduled in the link-frame under consideration. Figure 3.2 shows a network for which the link-precedence graph is a tree. Figure 3.3 shows a zero-wait schedule constructed for this network using A_{tree}.

Algorithm A_{tree} :

Step 1) Select a root vertex for the link-precedence tree. Number all $|L|$ vertices in the tree, in non-decreasing order of distance in link-hops from the root. Renumber links in the network so that each link-precedence graph vertex and its corresponding link have the same number.

Step 2) Construct an integer schedule for all priority-slots in the frame on link number 1.

Step 3) Perform the following $|L| - 1$ iterations. At iteration l, $2 \leq l \leq |L|$, construct an integer schedule for all priority-slots in the frame on the l-th link as follows.

There is an integer i, $1 \leq i \leq l - 1$, such that either (i, l) or (l, i) is an arc in the link-precedence graph. Suppose that (i, l) is an arc in the link-precedence graph. Then, the i-th link is received at the node from which the l-th link transmits. For each session that has the i-th and l-th links as consecutive links in its path, schedule its priority-slot on the l-th link so as to start when its priority-slot finishes on the i-th link, modulo the frame-time. Suppose, instead, that (l, i) is an arc in the link-precedence graph. Then, the l-th link is received at the

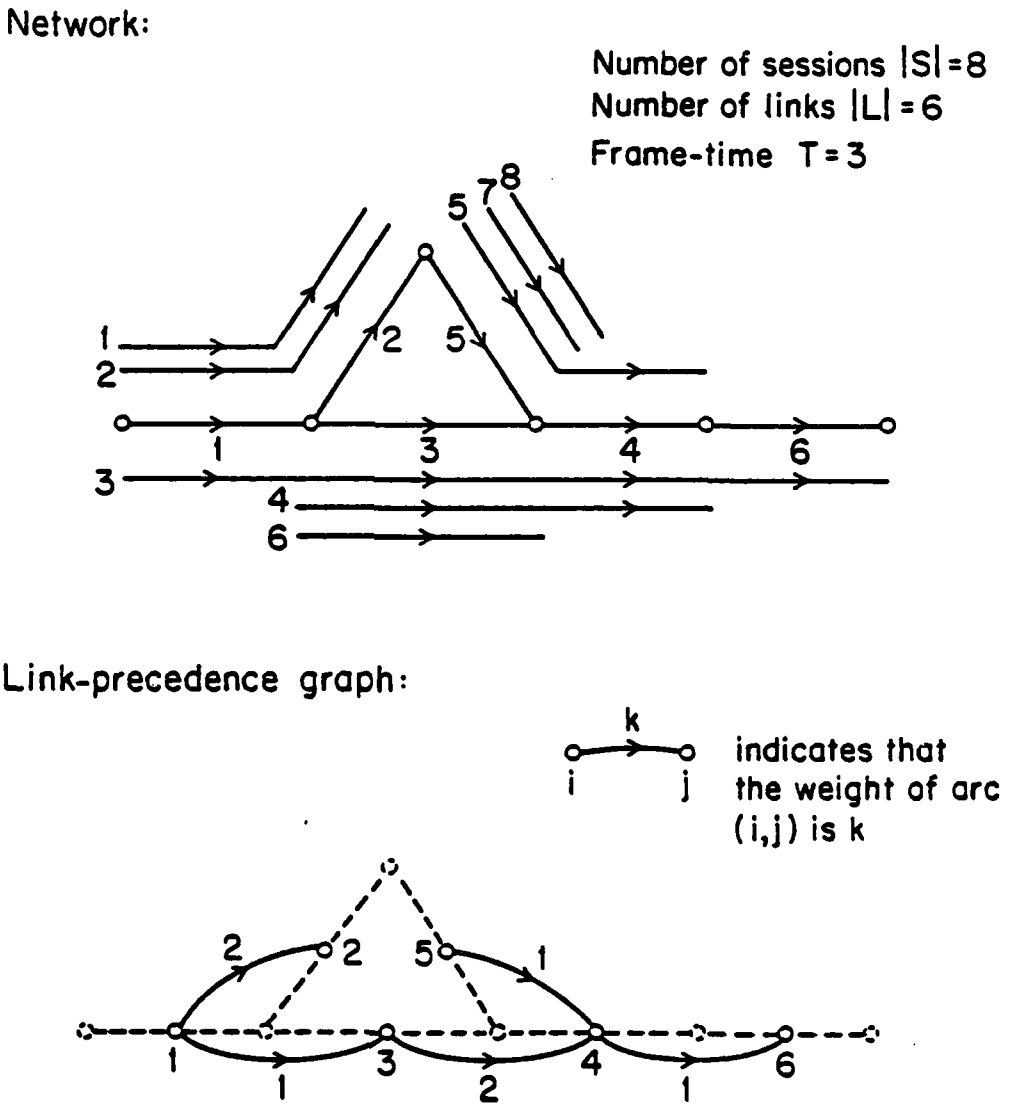


Figure 3.2 The Link-Precedence Graph for a Network

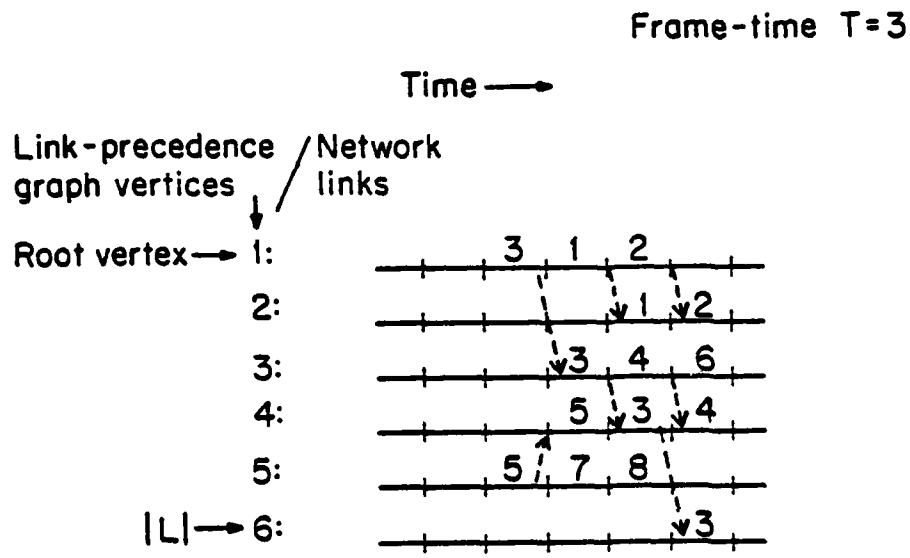


Figure 3.3 A Zero-Wait Schedule Constructed Using Algorithm A_{tree} for the Network of Figure 3.2

node from which the i-th link transmits. For each session that has the l-th and i-th links as consecutive links in its path, schedule its priority-slot on the l-th link so as to finish when its priority-slot starts on the i-th link, modulo the frame-time.

Suppose that some priority-slots remain to be scheduled in the frame on the l-th link. Since the link-precedence graph is a tree, these slots are the first of their respective sessions' priority-slots to be scheduled. Schedule these remaining priority-slots so as to start at integer times in the frame that are as yet unassigned.

This completes the description of A_{tree}.

The simplest non-trivial link-precedence graph that is not a tree is a directed triangle. In this case, as shown in Appendix D, the minimum sum of schedule-waits is 0 or 1, and there is a linear-time algorithm for constructing a minimum-wait integer schedule. If the link-precedence graph is a triangle with tree-offshoots, the following linear-time algorithm computes an integer schedule that has minimum sum of schedule-waits. First, all priority-slots that are to be scheduled on the links corresponding to the vertices of the triangle are scheduled using the algorithm for a triangular link-precedence graph. Then, each of the tree-offshoots is scheduled using algorithm A_{tree}, with the appropriate triangle vertex already scheduled chosen as root vertex. The minimum sum of schedule-waits is either 0 or 1.

Algorithm A_{tree} can be extended so as to apply to general networks. The extended algorithm $A_{heuristic}$, described below, is applied to each connected link-precedence graph component LPG.

Algorithm $A_{heuristic}$:

Step 1) Compute a maximum-weight spanning tree MST, for LPG. This can be done in quadratic-time by appropriately applying an algorithm, such as described by Papadimitriou and Steiglitz [27], for computing minimum-weight spanning trees in general graphs.

Step 2) Select a root vertex for MST. Number all $|L|$ vertices in MST in non-decreasing order of distance,in MST, in link-hops from the root. Renumber links in the network so that each link-precedence graph vertex and its corresponding link have the same number.

Step 3) Construct an integer schedule for all priority-slots in the frame on link number 1.

Step 4) Perform the following $|L| - 1$ iterations. At iteration l, $2 \leq l \leq |L|$, compute an integer schedule for all priority-slots in the frame on the l-th link as follows.

Let A_l denote the set of arcs a in LPG, that are of the form $a = (i,l)$ (or $a = (l,i)$), where $1 \leq i \leq l-1$. Let W'_a denote the following function of the schedule to be computed for the l-th link. If $a = (i,l)$, then W'_a is the sum, over all w_a sessions that have the i-th and l-th links as consecutive links in their paths, of the modulo-frame-time wait between the finish of the session's priority-slot on the i-th link and the start of its priority-slot on the l-th link. Otherwise, $a = (l,i)$, and W'_a is the sum, over all w_a sessions that have the l-th and i-th links as consecutive links in their paths, of the modulo-frame-time wait between the finish of the session's priority-slot on the i-th link and the start of its priority-slot on the i-th link. Assign integer start-times in the frame on the l-th link to the priority-slots for the sessions sharing the link, using an assignment algorithm that minimizes $\sum_{a \in A_l} W'_a$. This can be done in cubic time by applying an optimal assignment algorithm such as described in Papadimitriou and Steiglitz [27].

This completes the description of $A_{heuristic}$.

The schedule-wait for a session that contributes to the weights of arcs in the link-precedence graph component LPG is the sum, over all arcs $a = (i,l)$ in LPG such that the i-th and l-th links are consecutive links in the session's path, of the modulo-frame-time wait between the finish of the session's priority-slot on the i-th link and the start of its priority-slot on the l-th link. W'_a, $a = (i,l)$, is the sum, over all w_a sessions that have the i-th and l-th links as consecutive links in their paths, of the modulo-frame-time wait between the finish of

the session's priority-slot on the i-th link and the start of its priority-slot on the l-th link. The set of all arcs in LPG is the union of the $|L|-1$ sets A_l, $2 \leq l \leq |L|$. Hence, the sum, over all sessions that contribute to the weights of arcs in LPG, of schedule-waits for sessions is $\sum_{l=2}^{|L|} \sum_{a \in A_l} W'_a$.

The sum of schedule-waits W_σ, for the integer schedule σ computed by $A_{heuristic}$, is shown below to satisfy the following upper bounds. Let T denote the frame-time for the network. Then,

$$W_\sigma \leq \begin{cases} (T-1)\sum_{(i,j) \in MST} w_{i,j} & \text{(Tree bound);} \\ \left(\frac{T-1}{2}\right)\sum_{(i,j) \in LPG} w_{i,j} & \text{(Assignment bound).} \end{cases} \tag{3.4}$$

The Tree bound is obtained as follows. The modulo-frame-time wait between the priority-slots for a session that are on consecutive links in the session's path, in an integer schedule, is an integer between 0 and $T-1$. At iteration l in step 4 of $A_{heuristic}$, there is an integer i, $1 \leq i \leq l-1$, such that either (i,l) or (l,i) is an arc in MST. Suppose that priority-slots were scheduled at iteration l so that $W'_a = 0$ when $a = (i,l)$ (or $a = (l,i)$), as is done in algorithm A_{tree}. Then, defining A'_l to be $A_l - \{(i,l)\}$ (or $A_l - \{(l,i)\}$),

$$\sum_{a \in A_l} W'_a \leq \sum_{a \in A'_l} w_a (T-1). \tag{3.5}$$

This inequality must also hold for the actual scheduling at iteration l, for which the value of $\sum_{a \in A_l} W'_a$ is minimum. Summing inequality 3.5 over all iterations l, $2 \leq l \leq |L|$, the Tree bound results.

The Assignment bound is obtained next. In Appendix E, it is shown that the priority-slots at iteration l in step 4 of $A_{heuristic}$ can be scheduled so that

$$\sum_{a \in A_l} W'_a \leq \left(\frac{T-1}{2}\right) \sum_{a \in A_l} w_a. \tag{3.6}$$

This inequality must also hold for the actual scheduling at iteration l. Summing inequality 3.6 over all iterations l, $2 \leq l \leq |L|$, the Assignment bound results.

The Tree bound is zero when LPG is a tree, and σ is then a schedule with zero sum of schedule-waits. More generally, a zero-wait schedule is obtained when the link-precedence graph is a forest. The Tree bound is proportional to the sum of the weights of all arcs in LPG that are not in MST. Since MST is a maximum-weight spanning tree for LPG, the value of this sum is minimum among those for all spanning trees. The Assignment bound is equal to the expected value of the sum of schedule-waits that results from scheduling at random, as shown below. Suppose that the start-time of each priority-slot in each link-frame is uniformly distributed among the T integer time-instants in the frame, and is independent of the start-times of priority-slots in other link-frames. Then, the expected value of the modulo-frame-time wait, between the priority-slots for a session that are on consecutive links in the session's path, is $(T-1)/2$. Hence, the expected value of the resulting sum of schedule-waits equals the Assignment bound.

Any integer schedule can be locally redefined without increasing the sum of schedule-waits by rescheduling a link, given the schedule on all its neighbouring links, using an optimal assignment algorithm. Suppose that links are considered for rescheduling in turn, keeping the schedule unchanged if rescheduling would leave the sum of schedule-waits unchanged, and rescheduling otherwise. Then, each rescheduling decreases the sum of schedule-waits by at least one. Suppose also that this iterative procedure is continued until each link has been considered for rescheduling at least once since the last rescheduling. Then, the sum of schedule-waits $W_{\sigma'}$ for the resulting schedule σ' also satisfies the Assignment bound. This result is obvious if the procedure above is followed starting with a schedule computed using algorithm $A_{heuristic}$. Otherwise, this result can be obtained by verifying the following statements. The proof given in Appendix E for inequality 3.6 holds when, in 3.6, the set A_l is replaced by the set A''_l of all arcs in LPG that are of the form (i,l) (or (l,i)). Suppose that schedule σ' is used for purposes of defining W'_a. Then, since an optimal assignment algorithm has been used at each iteration, inequality 3.6, with A_l replaced by A''_l, holds for $1 \leq l \leq |L|$. Further, $W_{\sigma'} = (\sum_{l=1}^{|L|} \sum_{a \in A''_l} W'_a)/2$. Hence, from 3.6, $W_{\sigma'} \leq ((T-1)/2)(\sum_{l=1}^{|L|} \sum_{a \in A''_l} w_a)/2 = ((T-1)/2) \sum_{a \in LPG} w_a$, and $W_{\sigma'}$ satisfies the Assignment bound.

Chapter 4

Mean Packet Delays for Poisson Packet Generation Model: Simulation Results for Some Networks

4.1 Introduction

In contrast with the upper bound developed in Chapter 2 for the packet intra-network delay, mean packet delays observed in simulations of three networks are presented in this chapter. Session packets are generated in Poisson processes at known rates that are less than the throughput-guarantees. For purposes of comparison, the simulations are conducted with, first, the schedule-based scheme, and, second, a scheme that uses conventional end-to-end windowing and first-come-first-served transmission of packets at links. The simulation results are interpreted in the light of some analysis.

The simulation results suggest that low mean values of packet end-to-end delays, from generation at source to reception at destination, are obtained in the schedule-based scheme even with small, but non-zero, window-sizes. Lower values of schedule-delays result in lower mean values of packet intra-network delays, but do not necessarily change mean packet end-to-end delays. In contrast with the low sensitivities of mean end-to-end delays to changes in window-sizes for the schedule-based scheme, these sensitivities for conventional end-to-end windowing, with first-come first-served transmission of packets at links, are observed to be relatively large.

The simulator, SB, for the schedule-based scheme implements the network model of Section 2.1 and the token usage algorithm of Figure 2.1, with the following additional assumptions.

a) Packets are generated for each session in an independent Poisson process.

b) All links have integer-valued slot start-times.

c) The time taken by a session's real tokens (or, window-tokens), to return to the session's source node after reaching its destination node, is a constant number of slots equal to the number of links in the session's path.

d) When a packet belonging to a session is transmitted in a non-priority slot on a subsequent link in the session's path, a real token is selected if possible, rather than a fictitious token, to be carried by the packet over the link.

e) A cyclic order for the sessions sharing a link is defined for each link. When a slot starts on a link it is first determined whether the slot is a non-priority slot, i.e., whether i) no session has pre-assigned transmission priority in the slot, or ii) the session that has pre-assigned transmission priority in the slot has no packet to transmit. Then, if the slot is a non-priority slot, the cyclic order is used to search for a session that has both a packet to transmit and a token for the packet to carry over the link. The search is started beginning with the session in the cyclic order that follows the session for which a packet was transmitted in the previous non-priority slot.

The simulator, FCFS, for the scheme that uses conventional end-to-end windowing and first-come first-served transmission of packets at links, implements the network model of Section 2.1, with the additional assumptions a),b),c) above, and f),g) below.

f) Window-tokens for a session, equal in number to the session's window-size, are created at session set-up time, at the session's source node. A packet belonging to the session is required to acquire a window-token at the source node in order to be enqueued in the first-come first-served queue of packets at the first link in the session's path, and deposit the token at the session's destination node upon reception there. Packets generated at the source node wait there in a first-come first-served queue before joining the queue at the first link when tokens become available.

g) At the ends of slots, packets are enqueued in the first-come first-served queues of packets at links as follows. First, a packet just received over a link incoming to a node is

enqueued at the next link, if any, in its path. The incoming links are considered here in ascending order of the identity numbers of the links. Next, a window-token just returned to a session's source node is used to enqueue a packet, if any, waiting to be enqueued at the first link in its path. The session window-tokens are considered here in ascending order of the identity numbers of the sessions.

FORTRAN programs for simulators SB and FCFS are listed in Appendix G. In the following sections, simulation results are presented for three networks. The packet generation rate and the window-size for the s-th session in a network are denoted by λ_s and w^s, respectively.

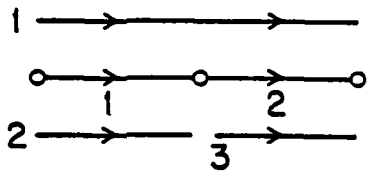
4.2 Network 1

Network 1 is shown in Figure 4.1. The window-sizes for all sessions are assumed equal to infinity.

From the analysis in Appendix F of a slotted link with Poisson packet arrivals, it follows that the expected packet waiting-time at link 1 for sessions 1 and 2 combined is $1/[2(1-\lambda_1-\lambda_2)]$, for both simulators SB and FCFS. With $\lambda_1 = \lambda_2 = 0.49$, this value is 25.

Suppose that $\lambda_1 = \lambda_2 = 0.5 + \epsilon > 0.5$, and $\lambda_3 < 0.5$. Since $\lambda_1 + \lambda_2 = 2\lambda_1 = 2\lambda_2 > 1$, packets for sessions 1 and 2 are always available for transmission on link 1. Thus, in simulator SB, packets for session 1 arrive at link 2 at the starts of slots for session 1 on link 2; the packet arrivals constitute a deterministic process with rate 0.5. In simulator FCFS, since packets for sessions 1 and 2 arrive at link 1 in Poisson processes with equal rates, the packets in distinct positions in the queue at link 1 belong to session 1 with probability 0.5, independent of one another. Hence, in simulator FCFS, packets for session 1 arrive at link 2 at the starts of slots on link 2 in a Bernoulli process with rate 0.5. From the analysis in Appendix F of a slotted link with Poisson arrivals combined with, first, deterministic arrivals, and, second, Bernoulli arrivals, the expected packet waiting-time at link 2 for sessions 1 and 3 combined is $2\lambda_3/(1-4\lambda_3{}^2)$ for simulator SB, and $3\lambda_3/(1-4\lambda_3{}^2)$ for simulator FCFS. The waiting-time

Network:



Window-size $w^s = \infty$, $s = 1,2,3$

Zero-wait schedule for simulator SB:

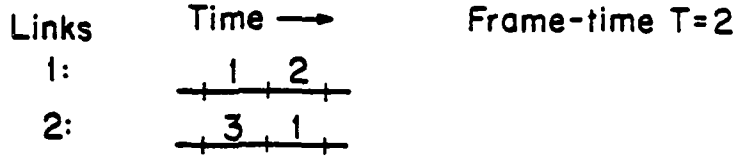


Figure 4.1 Network 1

for simulator SB is smaller because the arrival of packets for session 1 at link 2 is more regular. With $\lambda_3 = 0.25$, this value is 2/3 for simulator SB and 1 for simulator FCFS. The value with $\lambda_3 = 0.49$ is 24.75 for simulator SB and 37.12 for simulator FCFS. With link 1 loaded very heavily and link 2 loaded less heavily, using $\lambda_1 = \lambda_2 = 0.499$ and $\lambda_3 = 0.25$ for example, the results above provide approximate values for the expected waiting-time at link 2 for sessions 1 and 3 combined. Such estimates are not as accurate when the links are loaded equally heavily using $\lambda_1 = \lambda_2 = \lambda_3 = 0.49$, since the unused capacity $1 - (\lambda_1 + \lambda_3)$ on link 2 is then 0.02, as opposed to 0.01 when $\lambda_1 = \lambda_2 = 0.5$ and $\lambda_3 = 0.49$.

Simulation results are now presented.

a) Simulations with $\lambda_1 = \lambda_2 = 0.499$ and $\lambda_3 = 0.25$.

Mean packet waiting-times for sessions at link 2

as function of simulator

	Session 1	Session 3	1 and 3
SB	0.027	1.985	0.681
FCFS	0.872	1.247	0.997

The mean packet waiting-times for sessions 1 and 3 combined at link 2 agree with the corresponding analytical results.

b) Simulations with $\lambda_1 = \lambda_2 = \lambda_3 = \lambda$.

Mean packet waiting-times in simulator SB

at links, and overall, as function of λ

	Link 1	Link 2	Overall
$\lambda = 0.49$	25.14	19.12	29.51
$\lambda = 0.48$	12.43	9.48	14.60
$\lambda = 0.46$	6.20	4.70	7.27
$\lambda = 0.42$	3.12	2.32	3.62

Mean packet waiting-times in simulator FCFS

at links, and overall, as function of λ

	Link 1	Link 2	Overall
$\lambda = 0.49$	26.08	21.17	31.49
$\lambda = 0.48$	12.46	10.48	15.29
$\lambda = 0.46$	6.20	5.09	7.52
$\lambda = 0.42$	3.12	2.45	3.71

The data show that, with $\lambda = 0.49$, the mean packet waiting-times at link 2 in the simulators are lower than the respective approximate analytical estimates. The overall mean packet waiting-time for simulator FCFS is 6.5% higher than for simulator SB. Further, when λ is reduced to 0.42, this difference reduces to 2.5%. The simulation results indicate that, with infinite window-sizes, the overall mean packet waiting-time is less for simulator SB than for simulator FCFS. However, the difference is small except when links are loaded very close to their capacities.

4.3 Network 2

Network 2 is shown in Figure 4.2. Simulator SB can guarantee throughput up to 0.5 to each of the two sessions, irrespective of the session's window-size. A schedule with frame-time equal to 2 may be used for this purpose, with 1 slot per frame allotted to each session on each link in its path. Simulator FCFS cannot support a combined throughput of 1 when, for each session, the window-size equals the number of links in the session's path. This is first shown heuristically, and then demonstrated by simulations.

The round-trip time for a session's window-token is the time elapsed after the token is acquired by a packet at the session's source node until the token next returns to the source node. The maximum throughput possible with a single window-token is the reciprocal of its minimum round-trip time. For session 1, since the window-size is 3 and the round-trip time for a window-token is at least 6, the ratio of the window-size to the minimum round-trip time

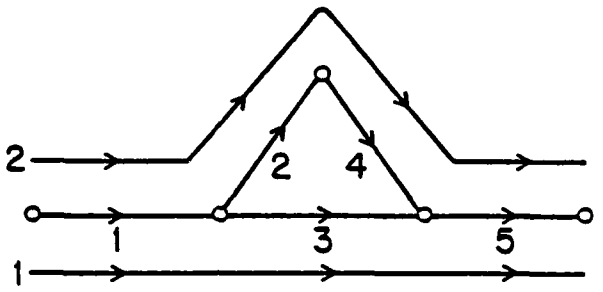


Figure 4.2 Network 2

is 0.5. Hence, the throughput for session 1 is at most 0.5. Similarly, the throughput for session 2 is at most 0.5.

Suppose that the combined throughput for sessions 1 and 2 equals 1. Then, every slot on link 1 must be used. A packet (numbered 1) belonging to session 1 must 'sometimes' be transmitted on link 1 immediately after a packet (numbered 2) belonging to session 2 is transmitted on the link. Then, packets 1 and 2 are transmitted concurrently on links 3 and 4, respectively. Since link 3 has a lower identity number than link 4, packet 1 is transmitted before packet 2 on link 5, and packet 2 must wait 1 time-unit at link 5. Hence, the round-trip time for a window-token for session 2 must 'sometimes' exceed 8, and the throughput for session 2 must be less than 0.5. If links 3 and 4 had their identity numbers interchanged. then packet 2 would have been transmitted before packet 1 on link 5, and the throughput for session 1 would have been less than 0.5. Thus, the combined throughput must be less than 1, and the supposition above is contradicted.

Simulation results are now presented for simulator FCFS, with window-sizes $w^1 = 3$ and $w^2 = 4$ as above and $\lambda_1 = \lambda_2 = \lambda$.

Mean window-token round-trip time for sessions
as function of λ
(and corresponding ratio of
window-size to mean round-trip time)

	Session 1	Session 2
$\lambda = 0.42$	6.30 (0.476)	9.48 (0.422)
$\lambda = 0.43$	6.26 (0.479)	9.53 (0.420)

Mean packet end-to-end delays for sessions
from generation at source to reception at destination
as function of λ

	Session 1	Session 2
$\lambda = 0.42$	8.84	251
$\lambda = 0.43$	9.55	∞^*

(* the throughput for session 2 is 0.42)

The data show that the mean window-token round-trip time for session 2 is significantly larger than the minimum value 8, and that the ratio of the window-size to the mean round-trip time is correspondingly less than 0.5. The simulation results indicate that simulator FCFS may not be able to support session rates approaching the throughput-guarantees of simulator SB, even if the window-size for each session is large enough to support the session's rate in the absence of other sessions.

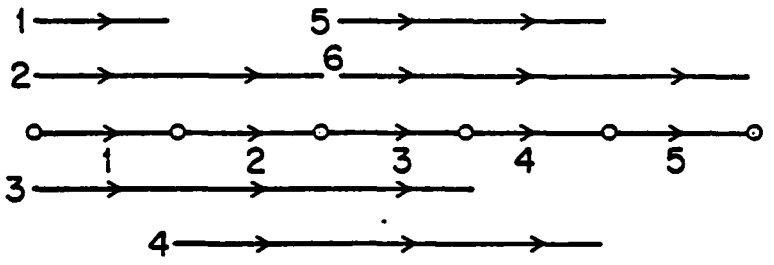
4.4 Network 3

Network 3 is shown in Figure 4.3. Mean packet end-to-end, pre-transmission, and intra-network waiting-times for sessions are measured in the simulations for this network.

The end-to-end waiting-time, for a packet belonging to a session that has H links in its path, is the difference between the end-to-end delay from generation at source to reception at destination, and the H time-units used for transmission of the packet. The pre-transmission waiting-time for the packet is the time from its generation to the start of its transmission on the first link in the path. The intra-network waiting-time for the packet is the difference between its end-to-end waiting-time and its pre-transmission waiting-time.

In simulator SB, if a session has window-size equal to zero, then it can transmit at most its throughput-guarantee of 1 packet per frame-time T on the first link in its path. The expected number of packets generated by the session in one frame-time is $T\lambda$. From the analysis

Network:

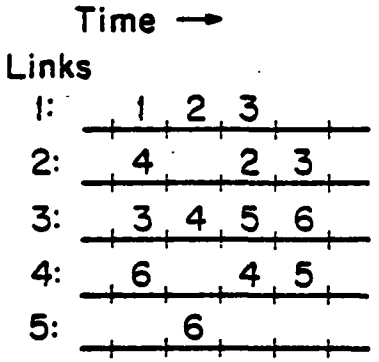


$\lambda_i = \lambda = 0{\cdot}24, i = 1,\ldots,6$

Schedules for simulator SB:

Frame-time T=4

Zero-wait schedule:



Maximum-wait schedule:

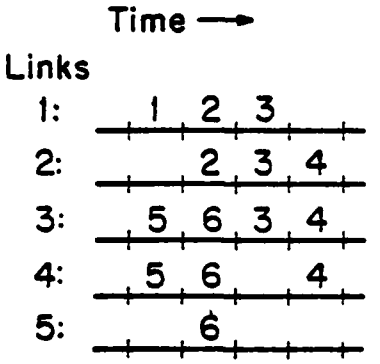


Figure 4.3 Network 3

in Appendix F for a slotted link with Poisson arrivals, it follows that the expected packet pre-transmission waiting-time for the session is $T/[2(1-T\lambda)]$, i.e., 50. When the zero-wait schedule is used, since the session has window-size equal to zero, the packet intra-network waiting-time for the session is zero and hence the expected packet end-to-end waiting-time also equals 50.

In both simulators SB and FCFS, when all window-sizes are infinite, the analysis in Appendix F for a slotted link with Poisson arrivals also provides the following approximation to the expected packet end-to-end waiting-time for a session. The approximation reduces the network to a single slotted link with Poisson packet arrivals at rate ρ, where ρ denotes the maximum, over all links in the session's path, of the sum of the packet generation rates for all sessions sharing a link. The expected packet end-to-end waiting-time for the session is approximately $1/[2(1-\rho)]$. The value of ρ is 0.72 for sessions 1 and 2 (the network is reduced to link 1 for these two sessions), and 0.96 for sessions 3,4,5, and 6 (the network is reduced to link 3 for these four sessions). Correspondingly, the expected packet end-to-end waiting-time is approximately 1.79 for sessions 1 and 2, and approximately 12.5 for sessions 3,4,5, and 6. It can also be verified that the end-to-end waiting-time for session 1 is 1.79, that for session 2 is greater than 1.79, and those for sessions 3,4,5, and 6 average to more than 12.5.

Simulation results are now presented. In each case, the network is simulated for one million time-units.

a) Simulations using simulator SB, with the zero-wait schedule and with equal window-size (1 or ∞) for all sessions.

Mean packet end-to-end waiting-times for sessions
as function of window-size w

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$w = 1$	2.14	3.22	15.00	15.58	15.01	14.60
$w = \infty$	1.79	2.33	12.71	13.29	12.96	12.53

The data for window-sizes equal to infinity are close in value to their respective analytical approximations. The value of the mean packet end-to-end waiting-time for a session when all

window-sizes are 1 is much closer to the corresponding value when all window-sizes are ∞ than to the value 50, that results when the session's window-size is 0. This suggests that low values of mean packet end-to-end delays are obtained for simulator SB even with small, but non-zero, window-sizes.

b) Simulations with window-sizes for sessions scaled in proportion to their path lengths.

Let k, $k = 1, 2, 3, 4, \infty$, denote the window-size scale-factor, i.e., $w^1 = k$, $w^2 = 2k$, $w^3 = 3k$, $w^4 = 3k$, $w^5 = 2k$ and $w^6 = 3k$.

i) First, mean packet end-to-end waiting-times are presented, for simulator SB with both the zero-wait and the maximum-wait schedules, and for simulator FCFS.

Mean packet end-to-end waiting-times for sessions
as function of k
in simulator SB with the zero-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$k = 1$	2.03	2.52	12.85	13.41	13.11	12.65
$k = 2$	1.82	2.38	12.72	13.29	12.96	12.53
$k = 3$	1.81	2.35	12.71	13.29	12.96	12.53
$k = 4$	1.80	2.34	12.71	13.29	12.96	12.53
$k = \infty$	1.79	2.33	12.71	13.29	12.96	12.53

Mean packet end-to-end waiting-times for sessions

as function of k

in simulator SB with the maximum-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$k = 1$	2.03	2.48	13.39	13.33	12.73	12.30
$k = 2$	1.82	2.40	13.24	13.32	12.63	12.26
$k = 3$	1.81	2.39	13.22	13.35	12.62	12.26
$k = 4$	1.80	2.39	13.22	13.36	12.61	12.26
$k = \infty$	1.79	2.39	13.22	13.37	12.61	12.25

Mean packet end-to-end waiting-times for sessions

in simulator FCFS as function of k

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$k = 1$	2.74	2.83	11.70	10.39	25.32	9.03
$k = 2$	1.86	2.35	11.58	10.05	20.82	8.80
$k = 3$	1.80	2.56	12.75	11.16	17.50	9.98
$k = 4$	1.78	2.64	13.50	11.90	15.25	10.77
$k = \infty$	1.78	2.70	14.61	13.02	11.91	11.91

The preceding data suggest that the dependence of mean packet end-to-end waiting-times on the positive integer window-size scale-factor is much weaker in simulator SB than in simulator FCFS. Further, for simulator SB, the difference between a session's mean packet end-to-end waiting-times with the zero-wait and the maximum-wait schedules is small. In simulator FCFS, the differences between the values of the data for sessions 5 and 6 are suggestive of high sensitivity to choice of window-size, as discussed further in part c).

ii) Next, mean packet intra-network and pre-transmission waiting-times are presented, for simulator SB with both the zero-wait and the maximum-wait schedules, and for simulator FCFS.

Mean packet intra-network waiting-times for sessions
as function of k
in simulator SB with the zero-wait schedule

	Session 2	Session 3	Session 4	1, 5, and 6
$k = 1$	0.52	5.45	5.33	0
$k = 2$	0.58	8.04	7.96	0
$k = 3$	0.58	9.30	9.32	0
$k = 4$	0.58	9.96	10.08	0
$k = \infty$	0.57	10.93	11.41	0

Mean packet intra-network waiting-times for sessions
as function of k
in simulator SB with the maximum-wait schedule

	Session 2	Session 3	Session 4	1, 5, and 6
$k = 1$	0.54	7.05	6.36	0
$k = 2$	0.61	9.08	8.66	0
$k = 3$	0.62	10.08	9.86	0
$k = 4$	0.62	10.62	10.55	0
$k = \infty$	0.63	11.44	11.74	0

Mean packet intra-network waiting-times for sessions

in simulator FCFS as function of k

	Session 2	Session 3	Session 4	1, 5, and 6
$k = 1$	0.33	3.19	2.90	0
$k = 2$	0.71	7.86	7.11	0
$k = 3$	0.85	10.25	9.31	0
$k = 4$	0.89	11.45	10.44	0
$k = \infty$	0.92	12.84	11.78	0

Mean packet pre-transmission waiting-times for sessions

as function of k

in simulator SB with the zero-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$k = 1$	2.03	2.00	7.40	8.08	13.11	12.65
$k = 2$	1.82	1.79	4.68	5.33	12.96	12.53
$k = 3$	1.81	1.77	3.41	3.97	12.96	12.53
$k = 4$	1.80	1.76	2.76	3.21	12.96	12.53
$k = \infty$	1.79	1.76	1.78	1.88	12.96	12.53

Mean packet pre-transmission waiting-times for sessions

as function of k

in simulator SB with the maximum-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$k = 1$	2.03	1.94	6.34	6.98	12.73	12.30
$k = 2$	1.82	1.79	4.17	4.66	12.63	12.26
$k = 3$	1.81	1.77	3.15	3.49	12.62	12.26
$k = 4$	1.80	1.76	2.60	2.81	12.61	12.25
$k = \infty$	1.79	1.76	1.78	1.63	12.61	12.25

Mean packet pre-transmission waiting-times for sessions

in simulator FCFS as function of k

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
$k = 1$	2.74	2.50	8.51	7.48	25.32	9.03
$k = 2$	1.86	1.64	3.72	2.93	20.82	8.80
$k = 3$	1.80	1.71	2.49	1.85	17.50	9.98
$k = 4$	1.78	1.75	2.05	1.47	15.25	10.77
$k = \infty$	1.78	1.78	1.78	1.24	11.91	11.91

The data show that the maximum-wait schedule indeed has the higher mean packet intra-network waiting-times. However, it also has the lower mean packet pre-transmission waiting-times, and its mean packet end-to-end waiting-times are essentially the same as a result. The mean packet intra-network waiting-times for sessions 2,3, and 4 when $k = 1$, i.e., when the window-size for each session equals the number of links in the session's path, are larger in simulator SB than in simulator FCFS. This may be explained as follows. For the same set of window-sizes, the number of window-tokens in simulator FCFS equals the number of real tokens in simulator SB. Since simulator SB has fictitious tokens in addition to real tokens, it is possible for a larger number of packets to be present inside the network in simulator SB than in simulator FCFS. The data indicate that this is indeed so for $k = 1$.

In simulator SB, a plausible explanation for the lower mean packet pre-transmission waiting-times for the maximum-wait schedule is as follows. With the maximum-wait schedule, at subsequent links in a session's path, packets belonging to the session have to wait longer for priority-slots, and hence have greater opportunity for using non-priority slots and carrying real tokens. This decreases the waiting-time for real tokens at the subsequent links, and increases the availability of real tokens at the first link. Hence, the session can use more non-priority slots on the first link, and its mean packet pre-transmission waiting-time is reduced. Further, since the session uses fewer priority-slots on a subsequent link, it creates more non-priority

slots on the link, and reduces the mean packet pre-transmission waiting-time for a session that has the link as its first link.

c).Simulations with separate increases in window-sizes for sessions.

The starting window-sizes are set at $w^1 = k,\ w^2 = 2k,\ w^3 = 3k,\ w^4 = 3k,\ w^5 = 2k,\ w^6 = 3k$, with the common window-size scale-factor k equal to 1 or 2. The window-size w^2 for session 2 and, separately, the window-size w^4 for session 4, are increased by the number of links in the respective session's path.

i) First, mean packet end-to-end, intra-network, and pre-transmission waiting-times are presented for simulator SB with both the zero-wait and the maximum-wait schedules, for $k = 1$, i.e., $(w^1, w^3, w^5, w^6) = (1,3,2,3)$.

Mean packet end-to-end waiting-times for sessions
as function of (w^2, w^4)
in simulator SB with the zero-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
(2,3)	2.03	2.52	12.85	13.41	13.11	12.65
(4,3)	2.01	2.46	12.85	13.40	13.11	12.65
(2,6)	2.03	2.51	12.84	13.35	13.09	12.64

Mean packet end-to-end waiting-times for sessions
as function of (w^2, w^4)
in simulator SB with the maximum-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
(2,3)	2.03	2.48	13.39	13.33	12.73	12.30
(4,3)	2.02	2.44	13.39	13.33	12.73	12.31
(2,6)	2.03	2.49	13.38	13.31	12.72	12.30

Mean packet intra-network waiting-times for sessions

as function of (w^2, w^4)

in simulator SB with the zero-wait schedule

	Session 2	Session 3	Session 4	1, 5, and 6
(2,3)	0.52	5.45	5.33	0
(4,3)	0.61	5.46	5.33	0
(2,6)	0.50	5.45	7.97	0

Mean packet intra-network waiting-times for sessions

as function of (w^2, w^4)

in simulator SB with the maximum-wait schedule

	Session 2	Session 3	Session 4	1, 5, and 6
(2,3)	0.54	7.05	6.36	0
(4,3)	0.59	7.05	6.36	0
(2,6)	0.55	7.04	8.67	0

Mean packet pre-transmission waiting-times for sessions

as function of (w^2, w^4)

in simulator SB with the zero-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
(2,3)	2.03	2.00	7.40	8.08	13.11	12.65
(4,3)	2.01	1.85	7.39	8.07	13.11	12.65
(2,6)	2.03	2.01	7.40	5.38	13.09	12.64

Mean packet pre-transmission waiting-times for sessions

as function of (w^2, w^4)

in simulator SB with the maximum-wait schedule

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
(2,3)	2.03	1.94	6.34	6.98	12.73	12.30
(4,3)	2.02	1.85	6.34	6.97	12.73	12.31
(2,6)	2.03	1.93	6.34	4.65	12.72	12.30

ii) Next, mean packet end-to-end waiting-times are presented for simulator FCFS, first for $k = 1$ and then for $k = 2$.

Mean packet end-to-end waiting-times for sessions

as function of (w^2, w^4)

in simulator FCFS for $(w^1, w^3, w^5, w^6) = (1, 3, 2, 3)$

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
(2,3)	2.74	2.83	11.70	10.39	25.32	9.03
(4,3)	3.35	1.69	12.54	10.53	24.99	8.90
(2,6)	2.68	3.41	15.79	4.38	25.92	10.08

Mean packet end-to-end waiting-times for sessions

as function of (w^2, w^4)

in simulator FCFS for $(w^1, w^3, w^5, w^6) = (2, 6, 4, 6)$

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
(4,6)	1.86	2.35	11.58	10.05	20.82	8.80
(6,6)	1.90	2.28	11.64	10.05	20.81	8.80
(4,9)	1.85	2.47	12.23	8.53	21.46	9.03

As in the simulations with window-sizes for all sessions increased simulataneously in proportion to their path-lengths, the data suggest that the sensitivity of a session's mean packet end-to-end waiting-time to an increase in its window-size is much smaller in magnitude in simulator SB than in simulator FCFS. Further, in simulator SB, the sensitivities of a session's mean packet end-to-end, intra-network, and pre-transmission waiting-times to an increase in a second session's window-size are much smaller in magnitude than the second session's respective sensitivities to the same increase. For mean packet end-to-end waiting-times in simulator FCFS, the two sets of sensitivities are often comparable in magnitude. This suggests that proper choice of window-sizes in simulator FCFS is more difficult than in simulator SB.

Chapter 5

Conclusion

An approach has been described for achieving packet throughput guarantees and packet intra-network delay limits for sessions in a network, while permitting flexible usage of link transmission capacities by the sessions. Sessions have packet throughput values up to which throughputs are guaranteed. Packet generation for sessions at rates above these values cannot always be supported, but, depending upon the level of inactivity of other sessions, may sometimes be carried with proper choice of window-sizes.

The schedule-based scheme in which session throughput guarantees equal to 1 packet per frame-time T are realized, has been explained in Chapter 2. The network model of Section 2.1 is used. The session's packet intra-network delay is upper-bounded by the sum of, first, the session's schedule-delay and, second, the product wT of the session's window-size w and the frame-time T. A similar upper bound can be obtained for the case of non-zero, but known and fixed, link propagation or nodal switching times, and also for the case of different slot durations that are required for links with different bit-speeds. The frame-time used on all links is the same, in either case.

A throughput value equal to n/T, where n is an integer at most equal to T, can be guaranteed to a session by assigning n priority-slots to the session in the frame on each link in the session's path. An improved delay-bound, equal to the sum of a suitably-defined schedule-delay with $\lceil w/n \rceil T$, would then appear to hold.

The network model of Section 2.1 conveys flow-control information to a session's source from its destination using end-to-end windowing. Node-by-node windowing can also be used for this purpose, as follows. Fictitious tokens for the session are defined as in Section 2.3, but real tokens for the session are associated with the nodes in the session's path. The token usage algorithm of Figure 2.1 can be used, with the understanding that a real token carried by a

packet over a link is associated with the node that receives the link. A real token associated with a node j in the session's path, is returned by node j to the node i preceding it in the path, when a packet belonging to the session is transmitted from node j to the node k following it in the path, using a real token associated with node k.

An alternative version of node-by-node windowing associates fictitious as well as real tokens with nodes in the session's path. A token carried by a packet over a link is associated with the node that receives the link. Fictitious tokens are used in priority-slots on the link, and real tokens in non-priority slots. If a fictitious token associated with a node j in the session's path is present at node j when a packet belonging to the session is transmitted from node j in a priority-slot, then the fictitious token is discarded at node j. Otherwise, a real token associated with node j is returned by node j to the node i preceding it in the path. When a packet belonging to the session is transmitted from node j in a non-priority slot, either a real token associated with node j is returned to the node i preceding it in the path, or a fictitious token associated with node j and present there is discarded there.

Algorithms for scheduling priority-slots so as to obtain low values of session schedule-delays have been presented in Chapter 3, following an NP-completeness proof showing that the minimum-delay scheduling problem is algorithmically hard for general networks. The network model of Section 2.1 is used, along with the assumption that the paths for all sessions in the network are known.

Algorithm A_{tree} constructs a schedule with sum of schedule-waits equal to zero, for networks with tree link-precedence graphs. For such networks, this algorithm can be extended so as to produce schedules that have waits between consecutive slots equal to known and fixed link propagation or nodal switching times.

Algorithm $A_{heuristic}$ is a scheduling heuristic that can be used for any network that conforms to the model assumed. This algorithm reduces to A_{tree} for tree link-precedence graphs. An analysis of its worst-case performance has been presented, but its performance for specific networks may need to be investigated further. $A_{heuristic}$ can be extended to the

non-zero link propagation or nodal switching time case, and also to the case of different slot durations on different links. Worst-case analysis for these extensions is likely to be complicated.

The schedule may require improvement during the course of operation of the network, as new sessions are set up and old ones disconnected. In order to reduce the time and communication required, the schedule may be computed using a distributed algorithm, where the computation and communication involving a link-precedence graph vertex are the joint responsibility of the transmitting and receiving nodes for the corresponding link.

A distributed implementation of algorithm $A_{heuristic}$ is possible. The distributed algorithm described by Gallager, Humblet, and Spira [11] would be used to compute a maximum-weight spanning tree and corresponding root vertex for each link-precedence graph component. Then, links would be scheduled using communication outwards along the link-precedence trees starting from the roots.

In order to improve the schedule, links may be rescheduled using an optimal assignment algorithm, holding fixed the priority-slot assignments on neighbouring links. The distribution of rescheduling opportunities among links can be done in the manner of the solution described by Chandy and Misra [6] for the distributed dining philosophers problem.

Packet intra-network delay limits may be violated if links are rescheduled while packets are in transit in the network, or if link speeds drift, and also if links are unreliable and packets have to be retransmitted. For example, the rate at which fictitious tokens for a session are carried over the first link in a session's path may not be matched by the fictitious token rate on subsequent links if these links are unreliable. In such a case, both real and fictitious tokens are backlogged at these links. The fictitious token backlog can be viewed as an effective increase in the real token window-size, and raises the packet intra-network delay.

When packets belonging to a session incur excessive intra-network delay, the session's source node may reduce the session's window-size by holding back real tokens. The window-size

may be increased when the delay decreases. The source node may also relieve excess intra-network delay for the session by temporarily blocking transmission of the session's packets on the first link in the session's path. Packet intra-network delay measurements for the session would be required, and the source node would need to be appropriately informed.

Mean packet delays observed in simulations of three networks, with session packets generated in Poisson processes at known rates that are less than the throughput guarantees, have been presented in Chapter 4. The simulator for the schedule-based scheme implements the network model of Section 2.1 and the token usage algorithm of Figure 2.1, with additional assumptions as listed in Section 4.1. The limited simulations suggest that, for Poisson packet generation at session rates less than throughput guarantees, low mean values of packet end-to-end delays, relatively insensitive to choice of window-sizes, are obtained even at small but non-zero window-sizes.

This hypothesis is conditional on the additional assumptions referred to above, and the extent of the dependence may need investigation. In particular, assumption (e), that round-robin discipline is used to allocate non-priority slots, may be important because this discipline should be fairer than first-come first-served discipline in offering opportunity of packet transmission to sessions (Hahne [17]). It is also likely that assumption (c), the use of fixed real token return times that assume zero waits for the tokens on their return paths in the network, reduces mean packet end-to-end delays. As mentioned in Section 4.4(b)(ii), assumption (d), the use of real tokens if possible rather than fictitious tokens in non-priority slots on subsequent links in the paths for sessions, may account for the lower mean packet pre-transmission waiting-times observed with schedules that have higher schedule-delays.

Large queueing delays are expected when session packets are generated at Poisson time-instants in batches, with geometric batch-size distribution for example (Fuchs and Jackson [8], Kekre, Saxena and Khalid [22]). Larger window-sizes may be required in simulations with such packet generation processes. Simulations with sets of session packet generation rates that exceed throughput guarantees but are feasible, may provide additional insight into the choice of

window-sizes. Simulations of schedule-based schemes using node-by-node windowing, instead of end-to-end windowing, may also provide further insight.

Appendix A

Proof of Upper Bound, of Section 2.4, for Packet Intra-Network Delay

Assume that the fictitious tokens for the session are carried in first-come first-served order over each link in the session's path, and that the first fictitious token has no fictitious tokens ahead of it in the path. Let $t_{f,h}$, $f \geq 1$, $1 \leq h \leq H$, denote the time at which a packet starts being transmitted on the h-th link carrying the f-th fictitious token. Priority-slots for the session start on the h-th link in the path, $1 \leq h \leq H$, at times $\sigma_h + nT$, for all integer values n. Since fictitious tokens are carried over the first link in priority-slots only, let the integer n_f be defined by

$$t_{f,1} = \sigma_1 + n_f T,\ f \geq 1,\ n_{f+1} \geq n_f + 1. \tag{A.1}$$

First, it is shown that the time $t_{1,h}$, $1 \leq h \leq H$, at which the first fictitious token is carried over the h-th link, satisfies

$$t_{1,h} \leq \sigma_h + n_1 T,\ 1 \leq h \leq H. \tag{A.2}$$

From A.1, $t_{1,1} = \sigma_1 + n_1 T$. For $2 \leq h \leq H$, assume, for purposes of induction, that $t_{1,h-1} \leq \sigma_{h-1} + n_1 T$. A priority-slot starts on the h-th link at time $\sigma_h + n_1 T$. Since $\sigma_{h-1} + n_1 T + 1 \leq \sigma_h + n_1 T$, this priority-slot starts after the first fictitious token has been carried over the $h-1$-th link. A fictitious token is used, if available, in such a slot. Therefore, $t_{1,h} \leq \sigma_h + n_1 T$. Thus, A.2 follows by induction.

Then, as shown below, the time $t_{f,h}$, $f \geq 1$, $1 \leq h \leq H$, at which the f-th fictitious token is carried over the h-th link, satisfies

$$t_{f,h} \leq \sigma_h + n_f T,\ f \geq 1,\ 1 \leq h \leq H. \tag{A.3}$$

For $f \geq 2$ and $2 \leq h \leq H$, assume, for purposes of induction, that $t_{f-1,h} \leq \sigma_h + n_{f-1} T$ and $t_{f,h-1} \leq \sigma_{h-1} + n_f T$. A priority-slot starts on the h-th link at time $\sigma_h + n_f T$. Since $\sigma_h + n_{f-1} T < \sigma_h + n_f T$ and $\sigma_{h-1} + n_f T + 1 \leq \sigma_h + n_f T$, this priority-slot starts after

the $f-1$-th and f-th fictitious tokens have been carried over the h-th and $h-1$-th links, respectively, and it follows that $t_{f,h} \leq \sigma_h + n_f T$. Since A.1 and A.2 hold, A.3 follows by induction.

Now, assume that a packet P starts being transmitted on the first link at time τ. Further, define F by $t_{F,1} \leq \tau < t_{F+1,1}$, i.e., at time τ, packet P either carries the F-th fictitious token over the first link, or carries a real token over the link, in between the times at which the F-th and $F+1$-th fictitious tokens are carried over the link.

Let Ω_h, $2 \leq h \leq H$, denote the set of fictitious tokens that, at time $\tau+1$, have been carried over the first link and are subsequently to be carried over the h-th link, i.e., $\Omega_h = \{f | 1 \leq f \leq F,\ t_{f,h} \geq \tau+1\}$, $2 \leq h \leq H$. From A.3, if $\sigma_h + n_F T < \tau+1$, then $t_{F,h} < \tau+1$, and $|\Omega_h| = 0$. Similarly, if $\sigma_h + n_F T \geq \tau+1$, then $|\Omega_h|$ is at most equal to the number of values of f, $1 \leq f \leq F$, for which $\sigma_h + n_f T \geq \tau+1$; since $\tau+1 = \sigma_h + \lceil n_F - (\sigma_h + n_F T - \tau - 1)/T \rceil T$, $\tau+1 \leq \sigma_h + \lceil n_F - \lfloor (\sigma_h + n_F T - \tau - 1)/T \rfloor \rceil T < \tau+1+T$, and hence $|\Omega_h| \leq 1 + \lfloor (\sigma_h + n_F T - \tau - 1)/T \rfloor$. Thus, for $2 \leq h \leq H$,

$$|\Omega_h| \begin{cases} = 0, & \text{if } \sigma_h + n_F T < \tau+1; \\ \leq 1 + \lfloor \frac{\sigma_h + n_F T - \tau - 1}{T} \rfloor, & \text{otherwise.} \end{cases} \qquad (A.4)$$

Number the packets for the session that are in the network at time $\tau+1$, assigning the serial number 1 to packet P, the next higher serial number to the next most recent packet to enter the network, and so on. Let p_h, $2 \leq h \leq H$, denote the highest serial number among the packets that, at time $\tau+1$, are subsequently to start being transmitted on the h-th link. Since the total number of real tokens for the session is the window-size w, $p_h \leq w + |\Omega_h|$, $2 \leq h \leq H$. Then, from A.4, $p_h \leq x_h$, $2 \leq h \leq H$, where x_h is defined as

$$x_h = \begin{cases} w, & \text{if } \sigma_h + n_F T < \tau+1; \\ w + 1 + \lfloor \frac{\sigma_h + n_F T - \tau - 1}{T} \rfloor, & \text{otherwise.} \end{cases} \qquad (A.5)$$

Packets are transmitted in order of decreasing serial number at each link. Let $u_{p,h}$, $2 \leq h \leq H$, $p_h \geq p \geq 1$, denote the time at which packet number p starts being transmitted on

the h-th link. The intra-network delay for the packet P, numbered 1, is $u_{1,H} - r + 1$, which is shown below to be upper-bounded by $(\sigma_H - \sigma_1 + 1) + wT$.

Let the integer n be defined by

$$r + 1 \leq \sigma_2 + nT < r + 1 + T. \tag{A.6}$$

Then, the first priority-slot for the session at or after time $r + 1$ on the second link starts at time $\sigma_2 + nT$. Since $\sigma_1 + n_F T \leq r$ by definition of F, $\sigma_2 + n_F T \leq r + T$, and it follows from A.6 that $n \geq n_F$. Further, from A.6,

$$\sigma_2 + n_F T \begin{cases} \geq r + 1, & \text{if } n = n_F; \\ < r + 1, & \text{if } n > n_F. \end{cases} \tag{A.7}$$

Let $s_{x,h}$, $2 \leq h \leq H$, $x_h \geq x \geq 1$, be defined as

$$s_{x,h} = \begin{cases} \sigma_h + nT + wT - (x-1)T, & \text{if } n = n_F; \\ \sigma_h + nT + (w-1)T - (x-1)T, & \text{if } n > n_F. \end{cases} \tag{A.8}$$

Then, a priority-slot for the session starts on the h-th link at each of the times $s_{x,h}$.

First, it is shown that packet number p, $p_h \geq p \geq p_{h-1} + 1$, $2 \leq h \leq H$, which at time $r + 1$ is to be subsequently transmitted first on the h-th link, starts being so transmitted at time $u_{p,h}$ that satisfies

$$u_{p,h} \leq s_{p,h}, \; 2 \leq h \leq H, \; p_h \geq p \geq p_{h-1} + 1, \tag{A.9}$$

where p_1 is defined as 0.

A priority-slot starts on the h-th link, $2 \leq h \leq H$, at time $s_{p_h,h}$. Since $p_h \leq x_h$, $s_{p_h,h} \geq s_{x_h,h}$, where $s_{x_h,h} \geq r+1$, as shown below. If $n = n_F$, then, from A.7, A.5, and A.8, for $2 \leq h \leq H$, $s_{x_h,h} = \sigma_h + n_F T + wT - (w + \lfloor \frac{\sigma_h + n_F T - r - 1}{T} \rfloor)T \geq r + 1$. If $n > n_F$, then, from A.7, there is an integer H', $2 \leq H' \leq H$, such that $\sigma_h + n_F T < r + 1$ for $2 \leq h \leq H'$ and $\sigma_h + n_F T \geq r + 1$ for $H' + 1 \leq h \leq H$. From A.5 and A.8, for $2 \leq h \leq H'$, $s_{x_h,h} = \sigma_h + nT \geq r + 1$; and for $H'+1 \leq h \leq H$, $s_{x_h,h} = \sigma_h + nT + (w-1)T - (w + \lfloor \frac{\sigma_h + n_F T - r - 1}{T} \rfloor)T \geq r + 1 + (n - n_F - 1)T \geq r + 1$. Thus, a priority-slot starts on the h-th link at time $s_{p_h,h}$, at or after time $r + 1$, and

it follows that $u_{p_h,h} \leq s_{p_h,h}$. For $p_h - 1 \geq p \geq p_{h-1} + 1$, assume, for purposes of induction, that $u_{p+1,h} \leq s_{p+1,h}$. A priority-slot starts on the h-th link at time $s_{p,h}$. Since $s_{p+1,h} < s_{p,h}$, the priority-slot starts after packet number $p+1$ has been transmitted on the h-th link, and it follows that $u_{p,h} \leq s_{p,h}$. Thus, A.9 follows by induction.

Then, as shown below, the time $u_{p,h}$, $2 \leq h \leq H$, $p_h \geq p \geq 1$, at which packet number p starts being transmitted on the h-th link, satisfies

$$u_{p,h} \leq s_{p,h},\ 2 \leq h \leq H,\ p_h \geq p \geq 1. \qquad (A.10)$$

For $3 \leq h \leq H$ and $p_{h-1} \geq p \geq 1$, assume, for purposes of induction, that $u_{p,h-1} \leq s_{p,h-1}$ and (for $p+1 \leq p_h$) $u_{p+1,h} \leq s_{p+1,h}$. A priority-slot starts on the h-th link at time $s_{p,h}$. Since $s_{p,h-1} + 1 \leq s_{p,h}$ and (for $p+1 \leq p_h$) $s_{p+1,h} < s_{p,h}$, this priority-slot starts after packet number p has been communicated over the $h-1$-th link and (for $p+1 \leq p_h$) packet number $p+1$ has been communicated over the h-th link. It follows that $u_{p,h} \leq s_{p,h}$. Since A.9 holds, A.10 follows by induction.

From A.10, the intra-network delay, $u_{1,H} - \tau + 1$, for packet P, is at most $s_{1,H} - \tau + 1$. If $n = n_F$, then, since $\sigma_1 + n_F T \leq \tau$ by definition of F, and from A.8, $s_{1,H} - \tau + 1 = (\sigma_h - \sigma_1 + 1) + wT + (\sigma_1 + n_F T - \tau) \leq (\sigma_H - \sigma_1 + 1) + wT$. If $n > n_F$, then, from A.8 and A.6, $s_{1,H} - \tau + 1 = (\sigma_H - \sigma_2 + 1) + wT + (\sigma_2 + nT - T - \tau) < (\sigma_H - \sigma_1 + 1) + wT$. Thus, the intra-network delay upper bound, $(\sigma_H - \sigma_1 + 1) + wT$, of Section 2.4, is proved.

Appendix B

A Procedure for Transforming from Non-Integer to Integer Schedules Without Increasing the Sum of Schedule-Delays

The procedure referred to in Section 3.1 is presented in this appendix.

Let x_i^0 denote the start-time of the i-th priority-slot in the given non-integer schedule. Assume that the priority-slots are numbered in non-decreasing order of $\epsilon_i^0 = x_i^0 - \lfloor x_i^0 \rfloor$, i.e., $\epsilon_i^0 \leq \epsilon_j^0$ for $i < j$. The sum of schedule-delays for the schedule can be expressed as $\sum_i c_i \epsilon_i^0 + c$. Here, c_i is -1 or 1 if the i-th priority-slot is on the first or last link, respectively, in the corresponding session's path, and the path has more than 1 link; c_i is 0 otherwise; and c is an integer constant.

Consider the following iterative procedure, starting with $k = 0$:

$$\epsilon_i^{k+1} = \begin{cases} \epsilon_i^k - \delta_{min}^k e_i^k, & \text{if } \sum_j c_j e_j^k \geq 0; \\ \epsilon_i^k + (1 - \delta_{max}^k) e_i^k, & \text{otherwise;} \end{cases}$$

where

$$e_i^k = \begin{cases} 0, & \text{if } \epsilon_i^k = 0 \text{ or } 1; \\ 1, & \text{if } 0 < \epsilon_i^k < 1; \end{cases}$$

$$\delta_{min}^k = \min_{i:e_i^k=1} \epsilon_i^k; \quad \delta_{max}^k = \max_{i:e_i^k=1} \epsilon_i^k.$$

The ϵ_i^k have the following properties: $0 \leq \epsilon_i^k \leq 1$; $\epsilon_i^k \leq \epsilon_j^k$ for $i < j$; for $k \geq 1$, $\sum_i c_i \epsilon_i^k + c \leq \sum_i c_i \epsilon_i^{k-1} + c$ and $\sum_i e_i^k \leq \sum_i e_i^{k-1} - 1$. The procedure terminates at $k = K$ where, for each i, ϵ_i^K is 0 or 1.

Let T denote the integer frame-time for the network. Define $x_i = \lfloor x_i^0 \rfloor + \epsilon_i^K$, $y_{i,m} = x_i + mT$, and $y_{i,m}^0 = x_i^0 + mT$, where m is an integer. Then, $y_{i,m} = y_{i,m}^0 - x_i^0 + x_i = \lfloor y_{i,m}^0 \rfloor - \lfloor x_i^0 \rfloor + x_i = \lfloor y_{i,m}^0 \rfloor + \epsilon_i^K$, and hence $\lfloor y_{i,m}^0 \rfloor \leq y_{i,m} \leq \lfloor y_{i,m}^0 \rfloor + 1$. Since $y_{i,m} \leq \lfloor y_{i,m}^0 \rfloor + 1$ and $\lfloor y_{j,n}^0 \rfloor \leq y_{j,n}$, if $y_{i,m}^0 + 1 \leq y_{j,n}^0$ and $\lfloor y_{i,m}^0 \rfloor + 1 \leq \lfloor y_{j,n}^0 \rfloor - 1$, then $y_{i,m} + 1 \leq y_{j,n}$. Since $\epsilon_i^0 = y_{i,m}^0 - \lfloor y_{i,m}^0 \rfloor$, if $y_{i,m}^0 + 1 \leq y_{j,n}^0$ and $\lfloor y_{i,m}^0 \rfloor + 1 = \lfloor y_{j,n}^0 \rfloor$, then $\epsilon_i^0 \leq \epsilon_j^0$; hence, $\epsilon_i^K \leq \epsilon_j^K$

and, since $y_{i,m} = \lfloor y_{i,m} \rfloor + \epsilon_i^K$, $y_{i,m} + 1 \leq y_{j,n}$. Thus, if $y_{i,m}^0 + 1 \leq y_{j,n}^0$ then $y_{i,m} + 1 \leq y_{j,n}$. Furthermore, $\sum_i c_i \epsilon_i^K + c \leq \sum_i c_i \epsilon_i^0 + c$. It follows that there exists an integer schedule for which x_i is the start-time of the i-th slot, and the sum of schedule-delays is not greater than that for the original non-integer schedule.

Appendix C

Corollaries of the NP-Completeness Result of Theorem 3.1

Consider a polynomial-time algorithm A for computing integer schedules. Let $A(I)$ denote the sum of schedule-waits for the schedule computed by algorithm A for a network instance I, and let $OPT(I)$ denote the minimum sum of schedule-waits for the instance. Assume that there are no polynomial-time algorithms for solving NP-complete problems. Then, the following results hold.

a) For any fixed positive integer K, there is an instance I such that $A(I) - OPT(I) > K$. This is shown as follows.

Suppose, to the contrary, that $A(I) - OPT(I) \leq K$ for all instances I. Consider the instance I' that consists of $K+1$ copies of an instance I. Then $OPT(I') = (K+1)OPT(I)$. The schedule computed by algorithm A for I' consists of a schedule σ_c for each copy c of I. Since $A(I') - OPT(I') \leq K$, $\sum_{c=1}^{K+1}[sum\ of\ schedule-waits\ for\ \sigma_c - OPT(I)] \leq K$, and therefore for at least one value of c the sum of schedule-waits for σ_c must be $OPT(I)$. This provides a polynomial-time algorithm for solving the 'Network 3-Periodic Zero-Wait Scheduling' problem, and Theorem 3.1 is contradicted. Hence, result a) is true.

b) For any fixed positive integer R, there is an instance I such that $A(I) > R\,OPT(I)$. This is shown as follows.

Suppose, to the contrary, that $A(I) \leq R\,OPT(I)$ for all instances I. Then, $OPT(I) = 0$ implies that $A(I) = 0$. If $A(I) = 0$ then $OPT(I) = 0$. Therefore, $OPT(I)$ is zero if, and only if, $A(I)$ is zero. This provides a polynomial-time algorithm for solving the 'Network 3-Periodic Zero-Wait Scheduling' problem, and Theorem 3.1 is contradicted. Hence, result b) holds.

c) For any fixed positive integers K and R, there is an instance I such that $OPT(I) \geq K$ and $A(I) > R\,OPT(I)$. This is shown as follows.

Suppose, to the contrary, that for all instances I such that $OPT(I) \geq K$, $A(I) \leq R\ OPT(I)$. Consider an instance I'' such that $OPT(I'') \geq K$ (I'' may consist of copies of any instance I''' such that $OPT(I''') \geq 1$). Consider the instance I' that consists of I'' and $R\ OPT(I'')$ copies of an instance I. Then, $OPT(I') \geq OPT(I'') \geq K$ and, hence, $A(I') \leq R\ OPT(I')$. The schedule computed by algorithm A for I' consists of a schedule for I'' and a schedule σ_c for each copy c of I. Suppose $OPT(I) = 0$. Then $OPT(I') = OPT(I'')$ and, hence, $A(I') \leq R\ OPT(I'')$. Then, since $OPT(I'') \geq K$, $K + \sum_{c=1}^{R\ OPT(I'')} (sum\ of\ schedule - waits\ for\ \sigma_c) \leq R\ OPT(I'')$. Therefore, for at least one value of c the sum of schedule-waits in σ_c must be 0. This provides a polynomial-time algorithm for solving the 'Network 3-Periodic Zero-Wait Scheduling' problem, and Theorem 3.1 is contradicted. Hence, result c) holds.

Appendix D

The Scheduling Algorithm, of Section 3.3, for Networks with Triangular Link-Precedence Graphs

Figure D.1 shows the type of network for which the link-precedence graph is a triangle. The number of sessions sharing any link is assumed to be at most equal to the frame-time T. The following scheduling algorithm constructs a minimum-wait integer schedule for such a network in linear-time. The minimum sum of schedule-waits is at most 1. Priority-slots are scheduled in the three link-frames, first for the two-link *ab*-, *bc*-, and *ca*-type sessions, and then for the single-link sessions.

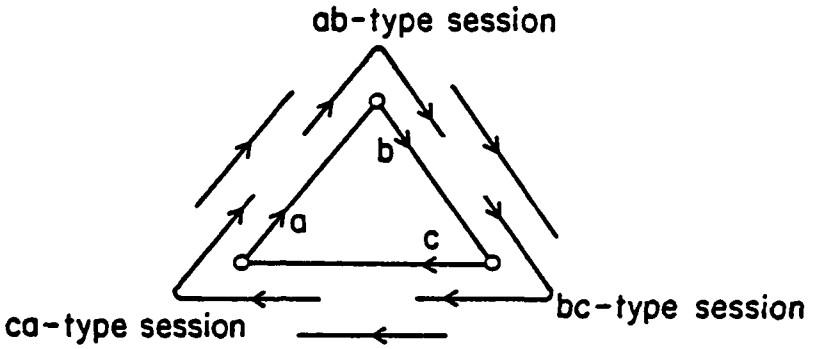
Let $w_{i,j}$ denote the weight of arc (i,j) in the link-precedence graph, i.e., $w_{i,j}$ is the number of ij-type two-link sessions using the i-th and j-th links. Assume, without loss of generality, that $w_{a,b} \leq w_{b,c}$ and $w_{a,b} \leq w_{c,a}$. Let $n_1 = w_{a,b}$, $n_2 = \min(w_{b,c}, w_{c,a}) - n_1$, and $n_3 = \max(w_{b,c}, w_{c,a}) - (n_1 + n_2)$. Several cases are defined below based on the values of n_1, n_2, and n_3. The sum of schedule-waits for the schedule that results is 0, except in case E(b) where it is 1.

Case A) $n_3 = 0$:

Schedule priority-slots for n_1 each of *ab*-, *bc*-, and *ca*-type sessions, and n_2 more each of *bc*- and *ca*-type sessions, as follows.

Link												
a:	*ab*	*ca*	...	...	*ab*	*ca*		*ca*		...		*ca*
b:	*bc*	*ab*	...	...	*bc*	*ab*	*bc*		...		*bc*	
c:	*ca*	*bc*	...	...	*ca*	*bc*	*ca*	*bc*	...	...	*ca*	*bc*
	1st	*1st*	...	...	n_1*th*	n_1*th*	*1st*	*1st*	...	...	n_2*th*	n_2*th*

Network:



Link-precedence graph:

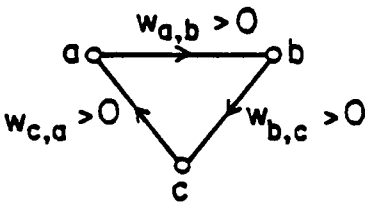


Figure D.1 Network and Link-Precedence Graph for Appendix D

Case B) $n_3 \geq 1,\ n_2 \geq 1$, and $w_{b,c} > w_{c,a}$:

Continue the schedule of case A as follows (in the sequel, a '*' indicates a priority-slot that had been scheduled previously, at either the same or a different time-instant in the frame).

Link							
a:		ca^*					
b:	bc^*	*bc*	*bc*	...	...	*bc*	
c:	ca^*	bc^*	*bc*	*bc*	...	...	*bc*
	n_2th	n_2th	*1st*	*2nd*	...	...	n_3th

Case C) $n_3 \geq 1,\ n_2 \geq 1$, and $w_{c,a} > w_{b,c}$:

Modify and continue the schedule of case A as follows.

Link							
a:		ca^*	*ca*	*ca*	...	...	*ca*
b:						bc^*	
c:	ca^*	*ca*	*ca*	...	...	*ca*	bc^*
		1st	*2nd*	...	...	n_3th	

(Priority-slots for the last of the n_2 *bc*-type sessions have been repositioned.)

Case D) $n_3 \geq 2,\ n_2 = 0$ (in the sequel, $w_{b,c} > w_{c,a}$, and the construction to be used if $w_{c,a} > w_{b,c}$ is similar to that shown for $w_{b,c} > w_{c,a}$):

Modify and continue the schedule of case A as follows.

Link							
a:		ca^*				ab^*	
b:	bc^*	*bc*	*bc*	...	...	*bc*	ab^*
c:	ca^*	bc^*	*bc*	*bc*	...	...	*bc*
	n_1th	n_1th	*1st*	*2nd*	...	...	th

(Priority-slots for the last of the n_1 ab-type sessions have been repositioned.)

Case E) $n_3 = 1,\ n_2 = 0$:

Sub-case a) $T \geq 2n_1 + 2$:

Continue the schedule of case A as follows.

Link				
a:	ab^*	ca^*		
b:	bc^*	ab^*	bc	
c:	ca^*	bc^*		bc
	$n_1 th$	$n_1 th$		$n_3 th$

Sub-case b) $n_1 = 1,4,7,10,\ldots$, and $T = 2n_1 + 1$ $(= 3,9,15,21,\ldots)$:

Modify and continue the schedule of case A as follows.

Link			
a:	ab^*	ca^*	
b:	bc^*	bc	ab^*
c:	ca^*	bc^*	bc

(A priority-slot for the last of the n_1 ab-type sessions has been repositioned. The sum of schedule-waits is 1.)

Sub-case c) $n_1 = 2,5,8,11,\ldots$, and $T = 2n_1 + 1$ $(= 5,11,17,23,\ldots)$:

The set of six columns marked by '*'-s below is included in the link-frames $(n_1 - 2)/3$ times.

Link												
a:	*ca*	*ca*	*ab*	*ab*	*ab*	*ca*	...	*ca*	*ca*	*ab*	*ab*	
b:	*bc*	*bc*	*bc*	*ab*	*ab*	*ab*	...	*bc*	*bc*	*bc*	*ab*	*ab*
c:	*ca*	*bc*	*bc*	*bc*	*ca*	*ca*	...	*ca*	*bc*	*bc*	*bc*	*ca*
	*	*	*	*	*	*						

Sub-case d) $n_1 = 3, 6, 9, 12, \ldots$, and $T = 2n_1 + 1$ $(= 7, 13, 19, 25, \ldots)$:

The set of six columns marked by '*'-s below is included in the link-frames $(n_1 - 3)/3$ times.

Link														
a:	*ab*	*ab*	*ab*	*ca*	*ca*	*ca*	...	*ab*	*ab*	*ab*	*ca*	*ca*	*ca*	
b:	*bc*	*ab*	*ab*	*ab*	*bc*	*bc*	...	*bc*	*ab*	*ab*	*ab*	*bc*	*bc*	*bc*
c:	*bc*	*bc*	*ca*	*ca*	*ca*	*bc*	...	*bc*	*bc*	*ca*	*ca*	*ca*	*bc*	*bc*
	*	*	*	*	*	*								

Appendix E

Proof of Inequality 3.6

Priority-slots are scheduled on the l-th link at iteration l in step 4 of algorithm $A_{heuristic}$ using an assignment algorithm that minimizes $\sum_{a \epsilon A_l} W'_a$. In this appendix, an assignment algorithm is described for scheduling these priority-slots at integer time-instants so that

$$\sum_{a \epsilon A_l} W'_a \leq \left(\frac{T-1}{2}\right) \sum_{a \epsilon A_l} w_a. \tag{3.6}$$

A session is defined as being in transit on the l-th link if i) for link numbers i and j, the session has the i-th, l-th, and j-th links as consecutive links in its path, i.e., the i-th link precedes and the j-th link follows the l-th link in the session's path, and ii) $(i,l) \epsilon A_l$ and $(l,j) \epsilon A_l$, i.e., both the i-th and j-th links have already been scheduled. Link-frames repeat at intervals of the frame-time T. Hence, for each of the session's priority-slots on the i-th link, there correspond i) a priority-slot to be scheduled for the session on the l-th link, starting from 0 to $T-1$ time-units after the finish of the slot on the i-th link, and ii) a priority-slot for the session on the j-th link, where the integer length S of the gap between the finish of the slot on the i-th link and the start of the slot on the j-th link satisfies the condition $1 \leq S \leq T$. The amount of wait that the session contributes to $\sum_{a \epsilon A_l} W'_a$ is the sum of, first, the modulo-T wait between the finish of its slot on the i-th link and the start of its slot on the l-th link, and, second, the modulo-T wait between the finish of its slot on the l-th link and the start of its slot on the j-th link. If the session's priority-slot on the l-th link is scheduled so as to lie in the gap defined above, then the contributed wait is the difference, $S-1$, between the gap-length S and the unit duration of the slot. Otherwise, the session's priority-slot on the l-th link is scheduled so as to lie outside the gap of length S, but so as to finish within $T-S$ time-units after the end of the gap; then, since a priority-slot starts on the j-th link a frame-time T after the end of the gap, the session's contributed wait is $T+S-1$.

A session is defined as finishing (respectively, starting) on the l-th link if i) for link number i (respectively, j), the session has the i-th and l-th links (respectively, the l-th and j-th links) as consecutive links in its path, ii) $(i,l)\epsilon A_l$ (respectively, $(l,j)\epsilon A_l$), and iii) the session is not in transit on the l-th link. vertex. The l-th link is preceded (respectively, followed) in the session's path by the i-th (respectively, j-th) link. For each of the session's priority-slots on the i-th (respectively, j-th) link, there corresponds a priority-slot to be scheduled for the session on the l-th link, starting (respectively, finishing) from 0 to $T-1$ time-units after (respectively, before) the finish (respectively, start) of the slot on the i-th (respectively, j-th) link. Thus, the session's contributed wait is an integer ranging from 0 to $T-1$.

First, priority-slots for sessions that are in transit or that start or finish on the l-th link are scheduled in the frame on the link at integer time-instants, using the assignment algorithm to be described. Next, priority-slots for all other sessions that use the link are scheduled at integer time-instants in the frame that are as yet unassigned.

Let n^{tr} denote the number of sessions that are in transit, and n^{sf} the number that start or finish, on the l-th link. Then, $n^{sf}+n^{tr} \leq T$ and $\sum_{a\epsilon A_l} w_a = n^{sf}+2n^{tr}$.

Case A) Assume that $n^{tr}=0$.

Schedule priority-slots for the n^{sf} start/finish sessions so that their individual contributed waits are upper-bounded by 0,1,2,..., $n^{sf}-1$, respectively.

Then,

$$\sum_{a\epsilon A_l} W'_a \leq \sum_{i=0}^{n^{sf}-1} i = n^{sf}.\left(\frac{n^{sf}-1}{2}\right) \leq n^{sf}\left(\frac{T-1}{2}\right) = \left(\frac{T-1}{2}\right)\sum_{a\epsilon A_l} w_a.$$

Case B) Assume that $n^{tr}>0$.

Let K denote the number of distinct gap-length values among the sessions in transit, and let n_k, $1 \leq k \leq K$, denote the number of sessions in transit that have gap-length S_k,

where $S_i < S_j$ for $1 \leq i < j \leq K$. Then, $\sum_{k=1}^{K} n_k = n^{tr}$. Let

$$m_k = \min\left(S_k - \sum_{i=1}^{k-1} m_i,\ n_k\right),\ 1 \leq k \leq K. \qquad (E.1)$$

Then, $\sum_{i=1}^{k} m_i \leq S_k,\ 1 \leq k \leq K$.

Sub-case B(I) Assume, further, that $n^{sf} \leq \min_{1 \leq k \leq K}(S_k - \sum_{i=1}^{k} m_i)$.

Then, $n^{sf} = 0$ if $m_k < n_k$ for at least one value of k. Consider the following scheduling algorithm.

Step 1) If $m_k = n_k$ for all values of k, then schedule priority-slots for the n^{sf} start/finish sessions so that their individual contributed waits are upper-bounded by 0,1,2,...,$n^{sf} - 1$, respectively.

Step 2) For each value of k from 1 to K, in increasing order of k, schedule priority-slots for m_k of the n_k transit sessions that have gap-length S_k within their respective gaps.

This can be done since, when m ($< m_k$) of the m_k slots have been scheduled, the total number of slots that have been scheduled is

$$n^{sf} + \sum_{i=1}^{k-1} m_i + m \leq \left(S_k - \sum_{i=1}^{k} m_i\right) + \sum_{i=1}^{k-1} m_i + m = S_k - (m_k - m) < S_k.$$

$n^{sf} + \sum_{i=1}^{k} m_k$ slots will have been scheduled by the end of this step.

Step 3) If there is a value k_1, $1 \leq k_1 \leq K$, such that $m_{k_1} < n_{k_1}$ and $m_k = n_k$ for $k_1 + 1 \leq k \leq K$, then do the following for each value of k from 1 to k_1. Schedule priority-slots for the remaining $n_k - m_k$ transit sessions that have gap-length S_k, at integer time-instants in the frame that are as yet unassigned.

All $n^{sf} + n^{tr}$ slots will have been scheduled by the end of this step.

Now, $\sum_{a \in A_t} W'_a$ is upper-bounded as follows.

Sub-case B(I)(a) Assume, also, that $m_k = n_k$ for all values of k, $1 \le k \le K$.

$$\sum_{a \in A_l} W'_a \le \sum_{i=0}^{n^{ef}-1} i + \sum_{k=1}^{K} n_k(S_k - 1)$$
$$\le n^{ef}\left(\frac{n^{ef}-1}{2}\right) + (T-1)\sum_{k=1}^{K} n_k$$
$$\le (n^{ef} + 2n^{tr})\left(\frac{T-1}{2}\right) = \left(\frac{T-1}{2}\right)\sum_{a \in A_l} w_a.$$

Sub-case B(I)(b) Assume, instead, that $m_{k_1} < n_{k_1}$ and $m_k = n_k$ for $k_1 + 1 \le k \le K$.

From E.1, $m_{k_1} = S_{k_1} - \sum_{i=1}^{k_1-1} m_i$, or $\sum_{i=1}^{k_1} m_i = S_{k_1}$; hence, $n^{ef} = 0$.

$$\sum_{a \in A_l} W'_a \le \sum_{k=1}^{K} m_k(S_k - 1) + \sum_{k=1}^{k_1}(n_k - m_k)(T + S_k - 1)$$
$$= \sum_{k=k_1+1}^{K} n_k(S_k - 1) + \sum_{k=1}^{k_1}[m_k(S_k - 1) + (n_k - m_k)(T + S_k - 1)]. \qquad (E.2)$$

Now,

$$\sum_{k=1}^{k_1}[m_k(S_k - 1) + (n_k - m_k)(T + S_k - 1)] = (T-1)\sum_{k=1}^{k_1} n_k + \sum_{k=1}^{k_1} n_k S_k - T\sum_{k=1}^{k_1} m_k$$
$$= (T-1)\sum_{k=1}^{k_1} n_k + \sum_{k=1}^{k_1} n_k S_k - T S_{k_1}$$
$$\le (T-1)\sum_{k=1}^{k_1} n_k - \left(T - \sum_{k=1}^{k_1} n_k\right) S_{k_1} \qquad (E.3)$$

Dropping the negative term in E.3, substituting the result in E.2, and then upper-bounding $S_k - 1$ by $T - 1$,

$$\sum_{a \in A_l} W'_a \le (T-1)\sum_{k=1}^{K} n_k = \left(\frac{T-1}{2}\right)\sum_{a \in A_l} w_a.$$

Sub-case B(II) Assume, instead, that $n^{ef} > \min_{1 \le k \le K}(S_k - \sum_{i=1}^{k} m_i)$.

Let $S_0 = 0$, $k_0 = 0$, and

$$k_j = \max \arg \min_{k_{j-1}+1 \le k \le K} \left(S_k - \sum_{i=1}^{k} m_i\right), \; 1 \le j \le J, \qquad (E.4)$$

where $S_{k_J} - \sum_{i=1}^{k_J} m_i < n^{ef} \leq \min_{k_J+1 \leq k \leq K}(S_k - \sum_{i=1}^{k} m_i)$ (or $k_J = K$ and $S_K - \sum_{i=1}^{K} m_i < n_{ef}$).

Step 1) For each value of j from 1 to J, in increasing order of j, do the following.

1(i) Schedule priority-slots for $S_{k_j} - S_{k_{j-1}} - \sum_{i=k_{j-1}+1}^{k_j} m_i$ start/finish sessions so that their individual contributed waits are upper-bounded by $S_{k_{j-1}}$, $S_{k_{j-1}}+1$, $S_{k_{j-1}}+2, \ldots$, $S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} m_i - 1$, respectively.

1(ii) For each value of k from $k_{j-1}+1$ to k_j, in increasing order of k, schedule priority-slots for m_k transit sessions that have gap-length S_k within their respective gaps.

It can be verified by induction that $S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} m_i$ slots will have been scheduled by the end of sub-step 1(i). The scheduling in sub-step 1(ii) can be done since, when m ($< m_k$) of the m_k slots have been scheduled, the total number of slots that have been scheduled is

$$\left(S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} m_i\right) + \sum_{i=k_{j-1}+1}^{k-1} m_i + m \leq \left(S_k - \sum_{i=k_{j-1}+1}^{k} m_i\right) + \sum_{i=k_{j-1}+1}^{k-1} m_i + m$$
$$= S_k - (m_k - m) < S_k.$$

Adding the $\sum_{i=k_{j-1}+1}^{k_j} m_i$ slots scheduled in sub-step 1(ii) to those scheduled in sub-step 1(i), a total of S_{k_j} slots will have been scheduled by the end of sub-step 1(ii).

$\sum_{i=1}^{k_J} m_i$ slots for transit sessions and $S_{k_J} - \sum_{i=1}^{k_J} m_i$ slots for start/finish sessions will have been scheduled by the end of this step.

Step 2) Schedule priority-slots for the remaining $n^{ef} - (S_{k_J} - \sum_{i=1}^{k_J} m_i)$ start/finish sessions so that their individual contributed waits are upper-bounded by S_{k_J}, $S_{k_J}+1$, $S_{k_J}+2, \ldots$, $n^{ef} + \sum_{i=1}^{k_J} m_i - 1$, respectively.

All n^{ef} start/finish slots will have been scheduled by the end of this step.

Step 3) For each value of k from k_J+1 to K, in increasing order of k, schedule priority-slots for m_k transit sessions that have gap-length S_k within their respective gaps.

This can be done since, when m ($< m_k$) of the m_k slots have been scheduled, the total number of slots that have been scheduled is

$$n^{sf} + \sum_{i=1}^{k-1} m_i + m \leq \left(S_k - \sum_{i=1}^{k} m_i \right) + \sum_{i=1}^{k-1} m_i + m$$
$$= S_k - (m_k - m) < S_k.$$

$n^{sf} + \sum_{i=1}^{K} m_k$ slots will have been scheduled by the end of this step.

Step 4) For each value of k from 1 to K, schedule priority-slots for the remaining $n_k - m_k$ transit sessions that have gap-length S_k at integer time-instants in the frame that are as yet unassigned.

All $n^{sf} + n^{tr}$ slots will have been scheduled by the end of this step.

Now, $\sum_{a \in A_l} W'_a$ is upper-bounded as follows.

Sub-case B(II)(a) Assume, also, that $m_k = n_k$ for all values of k, $1 \leq k \leq K$.

$$\sum_{a \in A_l} W'_a \leq \sum_{j=1}^{J} \left[S_{k_{j-1}} + (S_{k_{j-1}} + 1) + (S_{k_{j-1}} + 2) + \cdots + \left(S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} n_i - 1 \right) \right]$$
$$+ \sum_{j=1}^{J} \sum_{k=k_{j-1}+1}^{k_j} n_k (S_k - 1)$$
$$+ \left[S_{k_J} + (S_{k_J} + 1) + (S_{k_J} + 2) + \cdots + \left(n^{sf} + \sum_{i=1}^{k_J} n_i - 1 \right) \right]$$
$$+ \sum_{k=k_J+1}^{K} n_k (S_k - 1). \qquad (E.5)$$

The four terms in E.5 correspond to the waits contributed by substep 1(i), substep 1(ii), step 2, and step 3, respectively.

For $1 \leq j \leq J$,

$$\sum_{k=k_{j-1}+1}^{k_j} n_k (S_k - 1) \leq (S_{k_j} - 1) \sum_{k=k_{j-1}+1}^{k_j} n_k$$

$$= \left[\left(S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} n_i\right) + \left(S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} n_i + 1\right) + \cdots + (S_{k_j} - 1)\right]$$
$$+ \left[\left(\sum_{i=k_{j-1}+1}^{k_j} n_i - 1\right) + \left(\sum_{i=k_{j-1}+1}^{k_j} n_i - 2\right) + \cdots + 0\right]. \qquad (E.6)$$

This can be seen by adding corresponding terms of the two bracketed expressions above.

The first line in E.5 is a sum of increasing integers, with consecutive integers from $S_{k_{j-1}}$ to $S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} n_i - 1$ and then a gap between $S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} n_i - 1$ and S_{k_j}, for each value of j. The first bracketed expression in E.6 fills in the gap for the corresponding value of j. Thus, from E.5 and E.6,

$$\begin{aligned}
\sum_{a \in A_l} W'_a &\leq \left[1 + 2 + \cdots + \left(n^{ef} + \sum_{k=1}^{k_J} n_k - 1\right)\right] \\
&\quad + \sum_{j=1}^{J} \left[1 + 2 + \cdots + \left(\sum_{k=k_{j-1}+1}^{k_j} n_k - 1\right)\right] \\
&\quad + \sum_{k=k_J+1}^{K} n_k(S_k - 1) \\
&= \left(n^{ef} + \sum_{k=1}^{k_J} n_k\right)\left(\frac{n^{ef} + \sum_{k=1}^{k_J} n_k - 1}{2}\right) \\
&\quad + \sum_{j=1}^{J} \left(\sum_{k=k_{j-1}+1}^{k_j} n_k\right)\left(\frac{\sum_{k=k_{j-1}+1}^{k_j} n_k - 1}{2}\right) \\
&\quad + \sum_{k=k_J+1}^{K} n_k(S_k - 1) \\
&\leq \left(n^{ef} + \sum_{k=1}^{k_J} n_k\right)\left(\frac{T-1}{2}\right) + \left(\sum_{k=1}^{k_J} n_k\right)\left(\frac{T-1}{2}\right) + (T-1)\sum_{k=k_J+1}^{K} n_k \\
&= \left(n^{ef} + 2\sum_{k=1}^{K} n_k\right)\left(\frac{T-1}{2}\right) = \left(\frac{T-1}{2}\right)\sum_{a \in A_l} w_a.
\end{aligned}$$

Sub-case B(II)(b) Assume, instead, that $m_{k_1} < n_{k_1}$.

From E.1, $m_{k_1} = S_{k_1} - \sum_{i=1}^{k_1-1} m_i$, or $\sum_{i=1}^{k_1} m_i = S_{k_1}$; hence, from E.4, $m_k = n_k$ for $k_1 + 1 \le k \le K$. Thus, for $j = 1$ in step 1, no start/finish sessions are scheduled, and

$$\begin{aligned}\sum_{a \in A_l} W'_a \le & \sum_{k=1}^{k_1} m_k(S_k - 1) \\ & + \sum_{j=2}^{J} \left[S_{k_{j-1}} + (S_{k_{j-1}} + 1) + (S_{k_{j-1}} + 2) + \cdots + \left(S_{k_j} - \sum_{i=k_{j-1}+1}^{k_j} n_i - 1 \right) \right] \\ & + \sum_{j=2}^{J} \sum_{k=k_{j-1}+1}^{k_j} n_k(S_k - 1) \\ & + \left[S_{k_J} + (S_{k_J} + 1) + (S_{k_J} + 2) + \cdots + \left(n^{ef} + \sum_{i=1}^{k_J} m_i - 1 \right) \right] \\ & + \sum_{k=k_J+1}^{K} n_k(S_k - 1) + \sum_{k=1}^{k_1} (n_k - m_k)(T + S_k - 1). \end{aligned} \tag{E.7}$$

Combining the first and last terms by E.3, and combining the second, third, and fourth terms with the help of E.6,

$$\begin{aligned}\sum_{a \in A_l} W'_a \le & (T-1) \sum_{k=1}^{k_1} n_k - \left(T - \sum_{k=1}^{k_1} n_k \right) S_{k_1} \\ & + \left[S_{k_1} + (S_{k_1} + 1) + \cdots + \left(n^{ef} + \sum_{k=1}^{k_J} m_k - 1 \right) \right] \\ & + \sum_{j=2}^{J} \left[1 + 2 + \cdots + \left(\sum_{k=k_{j-1}+1}^{k_j} n_k - 1 \right) \right] \\ & + \sum_{k=k_J+1}^{K} n_k(S_k - 1). \end{aligned} \tag{E.8}$$

Since $\sum_{k=1}^{k_1} m_k = S_{k_1}$, and $m_k = n_k$ for $k_1 + 1 \le k \le k_J$,

$$\begin{aligned} & \left[S_{k_1} + (S_{k_1} + 1) + \cdots + \left(n^{ef} + \sum_{k=1}^{k_J} m_k - 1 \right) \right] \\ = & \left(n^{ef} + \sum_{k=k_1+1}^{k_J} n_k \right) S_{k_1} + \left[0 + 1 + \cdots + \left(n^{ef} + \sum_{k=k_1+1}^{k_J} n_k - 1 \right) \right]. \end{aligned} \tag{E.9}$$

This can be seen by adding S_{k_1} to each of the terms in the last bracketed expression above. Combining the second and third terms in E.8 with the help of E.9,

$$\begin{aligned}\sum_{a\epsilon A_i} W'_a \leq (T-1)\sum_{k=1}^{k_1} n_k - \left(T - n^{ef} - \sum_{k=1}^{k_J} n_k\right) S_{k_1} \\ + \left[1+2+\cdots+\left(n^{ef}+\sum_{k=k_1+1}^{k_J} n_k - 1\right)\right] \\ + \sum_{j=2}^{J}\left(\sum_{k=k_{j-1}+1}^{k_j} n_k\right)\left(\frac{\sum_{k=k_{j-1}+1}^{k_j} n_k - 1}{2}\right) \\ + \sum_{k=k_J+1}^{K} n_k(S_k - 1).\end{aligned}$$

Dropping the negative term in the above expression, and upper-bounding the third, fourth, and fifth terms,

$$\begin{aligned}\sum_{a\epsilon A_i} W'_a &\leq (T-1)\sum_{k=1}^{k_1} n_k + \left(n^{ef} + \sum_{k=k_1+1}^{k_J} n_k\right)\left(\frac{T-1}{2}\right) \\ &\quad + \left(\sum_{k=k_1+1}^{k_J} n_k\right)\left(\frac{T-1}{2}\right) + (T-1)\sum_{k=k_J+1}^{K} n_k \\ &= \left(n^{ef} + 2\sum_{k=1}^{K} n_k\right)\left(\frac{T-1}{2}\right) = \left(\frac{T-1}{2}\right)\sum_{a\epsilon A_i} w_a.\end{aligned}$$

Appendix F

The Expected Waiting-Times at a Slotted Link for Some Packet Arrival Processes

The analysis presented here is referred to in Chapter 4.

Assume that all slots are of unit duration, and that each packet requires one slot for transmission on the link. Three packet arrival processes are considered.

a) Poisson packet arrivals at rate ρ.

Let q_n denote the number of packets at the link at the beginning of the n-th slot, including the packet, if any, to be transmitted in the slot. Let P_ρ^n denote the number of packet arrivals during the n-th slot; P_ρ^n is a Poisson random variable with mean ρ. Then,

$$q_{n+1} = P_\rho^n + q_n - \begin{cases} 1, & \text{if } q_n \geq 1; \\ 0, & \text{if } q_n = 0. \end{cases} \qquad (F.1)$$

Let $G(z)$ denote the transform $E(z^{q_n})$ in steady-state. Then, transforming F.1,

$$G(z) = e^{\rho(z-1)}\{z^{-1}[G(z) - G(0)] + G(0)\}, \qquad (F.2)$$

so that

$$G(z) = G(0)\left[\frac{z-1}{ze^{-\rho(z-1)} - 1}\right], \qquad (F.3)$$

where $G(0)$ is the probability $1 - \rho$ that no packet is to be transmitted in a slot. $E(q_n)$ in steady-state is obtained from $G(z)$ as

$$E(q_n) = G'(1) = \rho + \frac{\rho^2}{2(1-\rho)}. \qquad (F.4)$$

The expected number of packets waiting at a random time is

$$E(q_n) - \frac{\rho}{2} = \frac{\rho}{2(1-\rho)}. \qquad (F.5)$$

Thus, from Little's theorem, the expected packet waiting-time is

$$\left(\frac{1}{\rho}\right)\frac{\rho}{2(1-\rho)} = \frac{1}{2(1-\rho)}. \qquad (F.6)$$

b) Poisson packet arrivals at rate λ, $0 \leq \lambda < 0.5$, combined with rate 0.5 deterministic packet arrivals, at the starts of alternate slots.

Assume that the deterministic packet arrivals are transmitted immediately upon arrival. Then, only Poisson packet arrivals may be required to wait. The Poisson arrivals are transmitted in alternate slots. Let type 'a' denote the type of these slots. Then, the number q_n of packets at the link, at the beginning of the n-th type 'a' slot, satisfies F.1 for $\rho = 2\lambda$. Hence, the expected number of packets waiting at a random time is obtained from F.5 as

$$\left[\frac{\rho}{2(1-\rho)}\right]_{\rho=2\lambda} = \frac{\lambda}{1-2\lambda}. \qquad (F.7)$$

Thus, from Little's theorem, the expected packet waiting-time is

$$\frac{1}{(\lambda+0.5)}\left(\frac{\lambda}{1-2\lambda}\right) = \frac{2\lambda}{1-4\lambda^2}. \qquad (F.8)$$

c) Poisson packet arrivals at rate λ, combined with rate p Bernoulli packet arrivals at the starts of slots. $0 \leq \lambda + p < 1$.

Let q_n denote the number of packets at the link at the beginning of the n-th slot, including the packet, if any, to be transmitted in the slot. Let P_λ^n and B_p^n denote the numbers of Poisson and Bernoulli packet arrivals, respectively, during the n-th slot; P_λ^n is a Poisson random variable with mean λ and B_p^n a Bernoulli random variable with mean p. Then,

$$q_{n+1} = P_\lambda^n + B_p^n + q_n - \begin{cases} 1, & \text{if } q_n \geq 1; \\ 0, & \text{if } q_n = 0. \end{cases} \qquad (F.9)$$

Let $G(z)$ denote the transform $E(z^{q_n})$ in steady-state. Then, transforming F.9,

$$G(z) = e^{\lambda(z-1)}[(1-p)+pz]\{z^{-1}[G(z)-G(0)]+G(0)\}, \qquad (F.10)$$

so that

$$G(z) = G(0)\left\{\frac{(z-1)[1+p(z-1)]}{ze^{-\lambda(z-1)} - [1+p(z-1)]}\right\}, \tag{F.11}$$

where $G(0)$ is the probability $1-(\lambda+p)$ that no packet is to be transmitted in a slot. $E(q_n)$ in steady-state is obtained from $G(z)$ as

$$E(q_n) = G'(1) = (\lambda+p) + \frac{\lambda(\lambda+2p)}{2[1-(\lambda+p)]}. \tag{F.12}$$

The expected number of packets waiting at a random time is

$$E(q_n) - (\frac{\lambda}{2}+p) = \frac{\lambda(1+p)}{2[1-(\lambda+p)]}. \tag{F.13}$$

Thus, from Little's theorem, the expected packet waiting-time, evaluated for $0 \leq \lambda < 0.5$ and $p = 0.5$, is

$$\left[\frac{\lambda(1+p)}{2[1-(\lambda+p)](\lambda+p)}\right]_{p=0.5} = \frac{3\lambda}{1-4\lambda^2}. \tag{F.14}$$

Appendix G

FORTRAN Programs for Simulators SB and FCFS

The programs listed in this appendix, and the respective starting page numbers, are as follows:

The programs are written in Data General's FORTRAN77 for the MV10000 computer. Comment statements in the programs begin with the symbol '!'. For ease of formatting the programs for inclusion in this appendix, continuation symbols have been omitted from continuation statements and statement labels have been shifted to the right.

```
Program SimulatorSB ! Main program for simulator SB

integer FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterTB.f77' ! TimeBufferLength
common / globalblock / NumberOfLinks, NumberOfSessions
integer Window, FinalTime
common / simulatorscheme / Window (MaxSessions)
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
real*8 Seed
common / arrivalblock / Seed, TotalGenerationRate,
   SessionDistribution (MaxSessions)
integer GenerationTimeBufferEntry, TransmissionTimeBufferEntry,
   TransmissionTimes, SumOfIntraNetworkDelays,
   SumOfSquaredIntraNetworkDelays, SumOfDelays,
   SumOfSquaredDelays
real MinEndToEndDelay, MaxEndToEndDelay,
   MinPreTransmissionDelay, MaxPreTransmissionDelay
real*8 GenerationTimes
common / measureblock / NumberInProcess (MaxSessions),
   GenerationTimeBufferEntry (MaxSessions),
   GenerationTimes (MaxSessions, TimeBufferLength),
   NumberGenerated (MaxSessions),
   TransmissionTimeBufferEntry (MaxSessions, MaxLinks),
   TransmissionTimes (MaxSessions, MaxLinks, TimeBufferLength),
   NumberTransmitted (MaxSessions, MaxLinks),
   MinEndToEndDelay (MaxSessions),
   MaxEndToEndDelay (MaxSessions),
   SumOfEndToEndDelays (MaxSessions),
   SumOfSquaredEndToEndDelays (MaxSessions),
   MinIntraNetworkDelay (MaxSessions),
   MaxIntraNetworkDelay (MaxSessions),
   SumOfIntraNetworkDelays (MaxSessions),
   SumOfSquaredIntraNetworkDelays (MaxSessions),
   MinPreTransmissionDelay (MaxSessions),
   MaxPreTransmissionDelay (MaxSessions),
   SumOfPreTransmissionDelays (MaxSessions),
   SumOfSquaredPreTransmissionDelay (MaxSessions),
   MinDelay (MaxSessions, MaxLinks),
   MaxDelay (MaxSessions, MaxLinks),
   SumOfDelays (MaxSessions, MaxLinks),
   SumOfSquaredDelays (MaxSessions, MaxLinks)
integer Session
real MeasuredGenerationRate (MaxSessions),
   MeanEndToEndDelay (MaxSessions),
   MeanSquaredEndToEndDelay (MaxSessions),
   MeanIntraNetworkDelay (MaxSessions),
   MeanSquaredIntraNetworkDelay (MaxSessions),
   MeanPreTransmissionDelay (MaxSessions),
   MeanSquaredPreTransmissionDelay (MaxSessions),
   MeasuredTransmissionRate (MaxSessions, MaxLinks),
   MeanDelay (MaxSessions, MaxLinks),
   MeanSquaredDelay (MaxSessions, MaxLinks)
```

```
dimension GenerationRate (MaxSessions)

do while (.true.)
   print *, 'Enter generation rates : '
   read *, (GenerationRate (Session),
      Session = 1, NumberOfSessions)
   TotalGenerationRate = 0.0
   do Session = 1, NumberOfSessions
      TotalGenerationRate = TotalGenerationRate +
         GenerationRate (Session)
   end do ! Total generation rate
   DistributionSum = 0.0
   SessionDistribution (NumberOfSessions) = 1.0
   do Session = 1, NumberOfSessions - 1
      DistributionSum = DistributionSum +
         GenerationRate (Session) / TotalGenerationRate
      SessionDistribution (Session) = DistributionSum
   end do ! Distribution function for session rates

   do while (.true.)
      print *, 'Enter windows : '
      read *, (Window (Session), Session = 1, NumberOfSessions)

      do while (.true.)
         print *, 'Enter Seed : ' ! Seed used : 314159.0
         read *, Seed
         InitialTime = 1

         do Session = 1, NumberOfSessions
            NumberInProcess (Session) = 0
            GenerationTimeBufferEntry (Session) = 0
            NumberGenerated (Session) = 0
            MinEndToEndDelay (Session) = 1.0E+9
            MaxEndToEndDelay (Session) = 0.0
            SumOfEndToEndDelays (Session) = 0.0
            SumOfSquaredEndToEndDelays (Session) = 0.0
            MinIntraNetworkDelay (Session) = 1000000000
            MaxIntraNetworkDelay (Session) = 0
            SumOfIntraNetworkDelays (Session) = 0
            SumOfSquaredIntraNetworkDelays (Session) = 0
            MinPreTransmissionDelay (Session) = 1.0E+9
            MaxPreTransmissionDelay (Session) = 0.0
            SumOfPreTransmissionDelays (Session) = 0.0
            SumOfSquaredPreTransmissionDelay (Session) = 0.0
            do i = 1, NumberOfLinksForSession (Session)
               Link = Links (Session, i)
               TransmissionTimeBufferEntry (Session, Link) = 0
               NumberTransmitted (Session, Link) = 0
               if (i .ge. 2) then
                  MinDelay (Session, Link) = 1000000000
                  MaxDelay (Session, Link) = 0
                  SumOfDelays (Session, Link) = 0
                  SumOfSquaredDelays (Session, Link) = 0
               end if
```

```
            end do
         end do ! Measure block initialization

      do while (.true.)
         print *, 'Enter FinalTime : '
         read *, FinalTime

! Simulate
         call ScheduleBasedScheme (InitialTime, FinalTime)

         RealFinalTime = Real (FinalTime)
         do Session = 1, NumberOfSessions
            MeasuredGenerationRate (Session) =
               Real (NumberGenerated (Session)) / RealFinalTime
            RealNumberProcessed =
               Real (NumberTransmitted (Session,
                  LastLink (Session)))
            MeanEndToEndDelay (Session) =
               SumOfEndToEndDelays (Session) /
                  RealNumberProcessed
            MeanSquaredEndToEndDelay (Session) =
               SumOfSquaredEndToEndDelays (Session) /
                  RealNumberProcessed
            MeanIntraNetworkDelay (Session) =
               Real (SumOfIntraNetworkDelays (Session)) /
                  RealNumberProcessed
            MeanSquaredIntraNetworkDelay (Session) =
               Real (SumOfSquaredIntraNetworkDelays (Session)) /
                  RealNumberProcessed
            MeanPreTransmissionDelay (Session) =
               SumOfPreTransmissionDelays (Session) /
                  RealNumberProcessed
            MeanSquaredPreTransmissionDelay (Session) =
               SumOfSquaredPreTransmissionDelay (Session) /
                  RealNumberProcessed
            do i = 1, NumberOfLinksForSession (Session)
               Link = Links (Session, i)
               MeasuredTransmissionRate (Session, Link) =
                  Real (NumberTransmitted (Session, Link)) /
                     RealFinalTime
               if (i .ge. 2) then
                  MeanDelay (Session, Link) =
                     Real (SumOfDelays (Session, Link)) /
                        RealNumberProcessed
                  MeanSquaredDelay (Session, Link) =
                     Real (SumOfSquaredDelays (Session, Link)) /
                        RealNumberProcessed
               end if
            end do
         end do ! Output-data computation

         do Session = 1, NumberOfSessions
            print *, 'Session = ', Session
            print *, 'MeasuredGenerationRate = ',
```

```
            MeasuredGenerationRate (Session)
         print *, 'MeanEndToEndDelay = ',
            MeanEndToEndDelay (Session)
         print *, 'MeanSquaredEndToEndDelay = ',
            MeanSquaredEndToEndDelay (Session)
         print *, 'MinEndToEndDelay = ',
            MinEndToEndDelay (Session)
         print *, 'MaxEndToEndDelay = ',
            MaxEndToEndDelay (Session)
         print *, 'MeanIntraNetworkDelay = ',
            MeanIntraNetworkDelay (Session)
         print *, 'MeanSquaredIntraNetworkDelay = ',
            MeanSquaredIntraNetworkDelay (Session)
         print *, 'MinIntraNetworkDelay = ',
            MinIntraNetworkDelay (Session)
         print *, 'MaxIntraNetworkDelay = ',
            MaxIntraNetworkDelay (Session)
         print *, 'MeanPreTransmissionDelay = ',
            MeanPreTransmissionDelay (Session)
         print *, 'MeanSquaredPreTransmissionDelay = ',
            MeanSquaredPreTransmissionDelay (Session)
         print *, 'MinPreTransmissionDelay = ',
            MinPreTransmissionDelay (Session)
         print *, 'MaxPreTransmissionDelay = ',
            MaxPreTransmissionDelay (Session)
         do i = 1, NumberOfLinksForSession (Session)
            Link = Links (Session, i)
            print *, 'Link = ', Link
            print *, 'MeasuredTransmissionRate = ',
               MeasuredTransmissionRate (Session, Link)
            if (i .ge. 2) then
               print *, 'MeanDelay = ',
                  MeanDelay (Session, Link)
               print *, 'MeanSquaredDelay = ',
                  MeanSquaredDelay (Session, Link)
               print *, 'MinDelay = ',
                  MinDelay (Session, Link)
               print *, 'MaxDelay = ',
                  MaxDelay (Session, Link)
            end if
         end do
      end do ! Output results

      print *, 'Current time = ', FinalTime
      print *, 'Enter "0" to exit run duration loop : '
      read *, Ind
      if (Ind .eq. 0) go to 40
      InitialTime = FinalTime + 1
   end do ! Run duration loop
40       continue
   print *, 'Enter "0" to exit seed loop : '
   read *, Ind
   if (Ind .eq. 0) go to 30
end do ! Seed loop
```

```
30              continue
      print *, 'Enter "0" to exit window loop : '
      read *, Ind
      if (Ind .eq. 0) go to 20
   end do ! Window loop
20          continue
   print *, 'Enter "0" to exit generation rate loop : '
   read *, Ind
   if (Ind .eq. 0) go to 10
end do ! Generation rate loop
10      continue
stop
end
```

```
! The core of simulator SB
Subroutine ScheduleBasedScheme (InitialTime, FinalTime)

integer FinalTime
integer WrapAroundIncrement, FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterP.f77' ! MaxPeriod
common / globalblock / NumberOfLinks, NumberOfSessions
integer Window
common / simulatorscheme / Window (MaxSessions)
integer Period, SessionFrame, TokenBufferLength, SessionCycle,
   OutLinks
common / schemeblock / Period,
   SessionFrame (MaxLinks, MaxPeriod),
   TokenBufferLength (MaxSessions),
   NumberOfSessionsOnLink (MaxLinks),
   SessionCycle (MaxLinks, MaxPeriod),
   OutLinks (MaxLinks, MaxSessions)
common / schemearrivals /
   NumberOfPackets (MaxLinks, MaxSessions)
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
integer Time, Slot, RoundRobinPosition (MaxLinks), Session,
   TokenBufferMarker (MaxSessions),
   TokenBuffers (MaxSessions, MaxLinks + 1), Packets, Tokens,
   TokenOut (MaxLinks), SessionServed (MaxLinks), Position,
   OutLink
dimension NumberOfTokens (MaxLinks, MaxSessions)
save

if (InitialTime .eq. 1) then ! Initialize
   Slot = 1
   do Link = 1, NumberOfLinks
      RoundRobinPosition (Link) = 1
   end do
   do Session = 1, NumberOfSessions
      TokenBufferMarker (Session) = 1
      do i = 1, TokenBufferLength (Session)
         TokenBuffers (Session, i) = 0
      end do
      Link = FirstLink (Session)
      NumberOfPackets (Link, Session) = 0
      NumberOfTokens (Link, Session) = Window (Session)
      do i = 2, NumberOfLinksForSession (Session)
         Link = Links (Session, i)
         NumberOfPackets (Link, Session) = 0
         NumberOfTokens (Link, Session) = 0
      end do
   end do ! Initializations complete
end if

do Time = InitialTime, FinalTime
   do Session = 1, NumberOfSessions
      Length = TokenBufferLength (Session)
```

```
      Marker = WrapAroundIncrement (TokenBufferMarker(Session),
         Length)
      TokenBufferMarker (Session) = Marker
      if (TokenBuffers (Session, Marker) .eq. 1) then ! Token
         Link = FirstLink (Session)
         NumberOfTokens (Link, Session) =
            NumberOfTokens (Link, Session) + 1
         TokenBuffers (Session, Marker) = 0
      end if
   end do ! Real tokens return to source nodes

   do Link = 1, NumberOfLinks
      Session = SessionFrame (Link, Slot)
      if (Session .gt. 0) then ! Session has priority to slot
         Packets = NumberOfPackets (Link, Session)
         if (Packets .gt. 0) then ! Session has packets waiting
            if (FirstLink (Session) .ne. Link) then ! In transit
               Tokens = NumberOfTokens (Link, Session)
               if (Tokens .eq. Packets) then ! Use real token
                  NumberOfTokens (Link, Session) = Tokens - 1
                  TokenOut (Link) = 1
               else ! Use fictitious token
                  TokenOut (Link) = 0
               end if
            else ! First link
               TokenOut (Link) = 0
            end if
            NumberOfPackets (Link, Session) = Packets - 1
            SessionServed (Link) = Session
            call Measure (Link, Session, 0.0, Time - 1)
            go to 10
         end if
      end if
      NumberInCycle = NumberOfSessionsOnLink (Link)
      Position = RoundRobinPosition (Link)
      do i = 1, NumberInCycle
         Position =
            WrapAroundIncrement (Position, NumberInCycle)
         Session = SessionCycle (Link, Position)
         Packets = NumberOfPackets (Link, Session)
         if (Packets .gt. 0) then ! Session has packets waiting
            Tokens = NumberOfTokens (Link, Session)
            if (FirstLink (Session) .ne. Link) then ! In transit
               go to 20
            else if (Tokens .gt. 0) then ! 1st link, real token
               go to 20
            end if
         end if
         if (i .eq. NumberInCycle) then
            SessionServed (Link) = 0
            go to 30
         end if
      end do ! Round-robin loop for searching in cyclic order
20          continue ! Serve session from round-robin
```

```
      if (Tokens .gt. 0) then ! Use real token
         NumberOfTokens (Link, Session) = Tokens - 1
         TokenOut (Link) = 1
      else ! Standard, in transit
         TokenOut (Link) = 0
      end if
      NumberOfPackets (Link, Session) = Packets - 1
      SessionServed (Link) = Session
      call Measure (Link, Session, 0.0, Time - 1)
30          continue
      RoundRobinPosition (Link) = Position
10          continue
   end do ! Link loop ends

! Generate packets in current slot
   call GenerateArrivals (Time)

   do InLink = 1, NumberOfLinks
      Session = SessionServed (InLink)
      if (Session .ne. 0) then ! Packet arrived
         OutLink = OutLinks (InLink, Session)
         if (OutLink .ne. 0) then ! Session in transit
            NumberOfPackets (OutLink, Session) =
               NumberOfPackets (OutLink, Session) + 1
            if (TokenOut (InLink) .eq. 1)
               NumberOfTokens (OutLink, Session) =
                  NumberOfTokens (OutLink, Session) + 1
         else ! Session exits
            if (TokenOut (InLink) .eq. 1) ! Return real token
               TokenBuffers (Session,
                  TokenBufferMarker (Session)) = 1
         end if
      end if
   end do ! In-link loop ends

   Slot = WrapAroundIncrement (Slot, Period) ! Next frame-slot
end do ! Time loop ends
return
end
```

```
! Generator of Poisson packets in slot for simulator SB
Subroutine GenerateArrivals (Time)

integer Time
integer FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
common / globalblock / NumberOfLinks, NumberOfSessions
real*8 Ran, Seed, GenerationTime
common / arrivalblock / Seed, TotalGenerationRate,
   SessionDistribution (MaxSessions)
common / schemearrivals /
   NumberOfPackets (MaxLinks, MaxSessions)
integer Session

RealTime = Real (Time - 1)
TimeGenerated = 0.0
do while (TimeGenerated .lt. 1.0) ! Generate next packet in slot
   TimeGenerated = TimeGenerated -
      Real (log (1.0 - Ran (Seed))) / TotalGenerationRate
   if (TimeGenerated .gt. 1.0) go to 10 ! No more packet in slot
   RandomValue = Real (Ran (Seed))
   do Session = 1, NumberOfSessions ! Generated packet's session
      if (RandomValue .le. SessionDistribution (Session)) then
         Link = FirstLink (Session)
         NumberOfPackets (Link, Session) =
            NumberOfPackets (Link, Session) + 1
         GenerationTime = RealTime + TimeGenerated
         call Measure (0, Session, GenerationTime, 0)
         go to 20
      end if
   end do ! Session loop ends
20      continue
end do ! Generation time loop ends
10      continue
return
end
```

```
! Recorder of measurements for simulator SB
Subroutine Measure (Link, Session, RealTimeMeasured,
   TimeMeasured)

integer Session, TimeMeasured
real*8 RealTimeMeasured
integer WrapAroundIncrement, FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterTB.f77' ! TimeBufferLength
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
integer GenerationTimeBufferEntry, TransmissionTimeBufferEntry,
   TransmissionTimes, SumOfIntraNetworkDelays,
   SumOfSquaredIntraNetworkDelays, SumOfDelays,
   SumOfSquaredDelays
real MinEndToEndDelay, MaxEndToEndDelay,
   MinPreTransmissionDelay, MaxPreTransmissionDelay
real*8 GenerationTimes
common / measureblock / NumberInProcess (MaxSessions),
   GenerationTimeBufferEntry (MaxSessions),
   GenerationTimes (MaxSessions, TimeBufferLength),
   NumberGenerated (MaxSessions),
   TransmissionTimeBufferEntry (MaxSessions, MaxLinks),
   TransmissionTimes (MaxSessions, MaxLinks, TimeBufferLength),
   NumberTransmitted (MaxSessions, MaxLinks),
   MinEndToEndDelay (MaxSessions),
   MaxEndToEndDelay (MaxSessions),
   SumOfEndToEndDelays (MaxSessions),
   SumOfSquaredEndToEndDelays (MaxSessions),
   MinIntraNetworkDelay (MaxSessions),
   MaxIntraNetworkDelay (MaxSessions),
   SumOfIntraNetworkDelays (MaxSessions),
   SumOfSquaredIntraNetworkDelays (MaxSessions),
   MinPreTransmissionDelay (MaxSessions),
   MaxPreTransmissionDelay (MaxSessions),
   SumOfPreTransmissionDelays (MaxSessions),
   SumOfSquaredPreTransmissionDelay (MaxSessions),
   MinDelay (MaxSessions, MaxLinks),
   MaxDelay (MaxSessions, MaxLinks),
   SumOfDelays (MaxSessions, MaxLinks),
   SumOfSquaredDelays (MaxSessions, MaxLinks)
integer Entry, OldTime, Delay
real*8 GenerationTime

if (Link .eq. 0) then ! Packet generated
   InProcess = NumberInProcess (Session)
   if (InProcess .eq. TimeBufferLength) then ! Overflow
      stop 'Time buffer overflow occurred'
   else ! Enter packet generated
      Entry = WrapAroundIncrement (GenerationTimeBufferEntry
         (Session), TimeBufferLength)
      GenerationTimeBufferEntry (Session) = Entry
      GenerationTimes (Session, Entry) = RealTimeMeasured
      NumberGenerated (Session) = NumberGenerated (Session) + 1
```

```
      NumberInProcess (Session) = InProcess + 1
   end if
else if (Link .ne. LastLink (Session)) then ! Enter transmission
   Entry = WrapAroundIncrement (TransmissionTimeBufferEntry
      (Session, Link), TimeBufferLength)
   TransmissionTimeBufferEntry (Session, Link) = Entry
   TransmissionTimes (Session, Link, Entry) = TimeMeasured
   NumberTransmitted (Session, Link) =
      NumberTransmitted (Session, Link) + 1
else ! Packet transmitted on last link in path
   Entry = WrapAroundIncrement (TransmissionTimeBufferEntry
      (Session, Link), TimeBufferLength)
   TransmissionTimeBufferEntry (Session, Link) = Entry
   TransmissionTimes (Session, Link, Entry) = TimeMeasured
   NumberTransmitted (Session, Link) =
      NumberTransmitted (Session, Link) + 1
   GenerationTime = GenerationTimes (Session, Entry)
   EndToEndDelay = TimeMeasured - GenerationTime + 1.0
   MinEndToEndDelay (Session) =
      Min (MinEndToEndDelay (Session), EndToEndDelay)
   MaxEndToEndDelay (Session) =
      Max (MaxEndToEndDelay (Session), EndToEndDelay)
   SumOfEndToEndDelays (Session) =
      SumOfEndToEndDelays (Session) + EndToEndDelay
   SumOfSquaredEndToEndDelays (Session) =
      SumOfSquaredEndToEndDelays (Session) +
      EndToEndDelay * EndToEndDelay
   OldTime = TransmissionTimes (Session, FirstLink (Session),
      Entry)
   IntraNetworkDelay = TimeMeasured - OldTime + 1
   MinIntraNetworkDelay (Session) =
      Min (MinIntraNetworkDelay (Session), IntraNetworkDelay)
   MaxIntraNetworkDelay (Session) =
      Max (MaxIntraNetworkDelay (Session), IntraNetworkDelay)
   SumOfIntraNetworkDelays (Session) =
      SumOfIntraNetworkDelays (Session) + IntraNetworkDelay
   SumOfSquaredIntraNetworkDelays (Session) =
      SumOfSquaredIntraNetworkDelays (Session) +
      IntraNetworkDelay * IntraNetworkDelay
   PreTransmissionDelay =
      EndToEndDelay - Real (IntraNetworkDelay)
   MinPreTransmissionDelay (Session) =
      Min (MinPreTransmissionDelay (Session),
      PreTransmissionDelay)
   MaxPreTransmissionDelay (Session) =
      Max (MaxPreTransmissionDelay (Session),
      PreTransmissionDelay)
   SumOfPreTransmissionDelays (Session) =
      SumOfPreTransmissionDelays (Session) +
      PreTransmissionDelay
   SumOfSquaredPreTransmissionDelay (Session) =
      SumOfSquaredPreTransmissionDelay (Session) +
      PreTransmissionDelay * PreTransmissionDelay
   do i = 2, NumberOfLinksForSession (Session)
```

```
      NewLink = Links (Session, i)
      NewTime = TransmissionTimes (Session, NewLink, Entry)
      Delay = NewTime - OldTime
      MinDelay (Session, NewLink) =
         Min (MinDelay (Session, NewLink), Delay)
      MaxDelay (Session, NewLink) =
         Max (MaxDelay (Session, NewLink), Delay)
      SumOfDelays (Session, NewLink) =
         SumOfDelays (Session, NewLink) + Delay
      SumOfSquaredDelays (Session, NewLink) =
         SumOfSquaredDelays (Session, NewLink) + Delay * Delay
      OldTime = NewTime
   end do ! Link delay loop ends
   NumberInProcess (Session) = NumberInProcess (Session) - 1
end if
return
end
```

```
! Functions used by both simulators SB and FCFS


Integer Function WrapAroundIncrement (Value, MaxValue)

integer Value
if (Value .lt. MaxValue) then ! Increment
   WrapAroundIncrement = Value + 1
else if (Value .eq. MaxValue) then ! Wrap around
   WrapAroundIncrement = 1
else ! Error
   WrapAroundIncrement = Value
    stop 'Invalid argument to WrapAroundIncrement'
end if
return
end


Integer Function FirstLink (Session)

integer Session
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)

FirstLink = Links (Session, 1)
return
end


Integer Function LastLink (Session)

integer Session
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)

LastLink = Links (Session, NumberOfLinksForSession (Session))
return
end
```

```
! Generator of uniform random numbers in [0,1),
! used by both simulators SB and FCFS
Real*8 Function Ran (Seed)

real*8 Seed, Combination, Modulus
real Multiplier
parameter (Modulus = 4294967296.0, Multiplier = 69069.0,
   Constant = 1.0)

Combination = Multiplier * Seed + Constant
if (Combination .gt. Modulus) then
   Seed = mod (Combination, Modulus)
else if (Combination .lt. Modulus) then
   Seed = Combination
else
   Seed = 0.0
end if
Ran = Seed / Modulus
return
end
```

```
! Data used by simulator SB for network of Figure 4.1
Block Data Network

include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterP.f77' ! MaxPeriod
common / globalblock / NumberOfLinks, NumberOfSessions
integer Period, SessionFrame, TokenBufferLength, SessionCycle,
   OutLinks
common / schemeblock / Period,
   SessionFrame (MaxLinks, MaxPeriod),
   TokenBufferLength (MaxSessions),
   NumberOfSessionsOnLink (MaxLinks),
   SessionCycle (MaxLinks, MaxPeriod),
   OutLinks (MaxLinks, MaxSessions)
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
data NumberOfLinks, NumberOfSessions / 2, 3 /
data Period / 2 /, SessionFrame (1,1) / 1 /,
   SessionFrame (1,2) / 2 /, SessionFrame (2,1) / 1 /,
   SessionFrame (2,2) / 3 /
data NumberOfSessionsOnLink (1) / 2 /,
   NumberOfSessionsOnLink (2) / 2 /
data SessionCycle (1,1) / 1 /, SessionCycle (1,2) / 2 /,
   SessionCycle (2,1) / 1 /, SessionCycle (2,2) / 3 /
data OutLinks (1,1) / 2 /, OutLinks (1,2) / 0 /,
   OutLinks (2,1) / 0 /, OutLinks (2,3) / 0 /
data NumberOfLinksForSession (1) / 2 /,
   NumberOfLinksForSession (2) / 1 /,
   NumberOfLinksForSession (3) / 1 /
data Links (1,1) / 1 /, Links (1,2) / 2 /, Links (2,1) / 1 /,
   Links (3,1) / 2 /
data TokenBufferLength (1) / 3 /, TokenBufferLength (2) / 2 /,
   TokenBufferLength (3) / 2 /
end
```

```
Program SimulatorFCFS ! Main program for simulator FCFS

integer FirstLink, LastLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterTB.f77' ! TimeBufferLength
common / globalblock / NumberOfLinks, NumberOfSessions
integer Window, FinalTime
common / simulatorscheme / Window (MaxSessions)
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
real*8 Seed
common / arrivalblock / Seed, TotalGenerationRate,
   SessionDistribution (MaxSessions)
integer GenerationTimeBufferEntry, WindowingTimeBufferEntry,
   TransmissionTimeBufferEntry, TransmissionTimes,
   SumOfIntraNetworkDelays, SumOfSquaredIntraNetworkDelays,
   SumOfDelays, SumOfSquaredDelays
real MinEndToEndDelay, MaxEndToEndDelay, MinIntraWindowDelay,
   MaxIntraWindowDelay, MinPreWindowDelay, MaxPreWindowDelay,
   MinPreTransmissionDelay, MaxPreTransmissionDelay,
   MinWindowToNetDelay, MaxWindowToNetDelay
real*8 GenerationTimes, WindowingTimes
common / measureblock / NumberInProcess (MaxSessions),
   GenerationTimeBufferEntry (MaxSessions),
   GenerationTimes (MaxSessions, TimeBufferLength),
   NumberGenerated (MaxSessions),
   WindowingTimeBufferEntry (MaxSessions),
   WindowingTimes (MaxSessions, TimeBufferLength),
   NumberWindowed (MaxSessions),
   TransmissionTimeBufferEntry (MaxSessions, MaxLinks),
   TransmissionTimes (MaxSessions, MaxLinks, TimeBufferLength),
   NumberTransmitted (MaxSessions, MaxLinks),
   MinEndToEndDelay (MaxSessions),
   MaxEndToEndDelay (MaxSessions),
   SumOfEndToEndDelays (MaxSessions),
   SumOfSquaredEndToEndDelays (MaxSessions),
   MinIntraWindowDelay (MaxSessions),
   MaxIntraWindowDelay (MaxSessions),
   SumOfIntraWindowDelays (MaxSessions),
   SumOfSquaredIntraWindowDelays (MaxSessions),
   MinPreWindowDelay (MaxSessions),
   MaxPreWindowDelay (MaxSessions),
   SumOfPreWindowDelays (MaxSessions),
   SumOfSquaredPreWindowDelays (MaxSessions),
   MinIntraNetworkDelay (MaxSessions),
   MaxIntraNetworkDelay (MaxSessions),
   SumOfIntraNetworkDelays (MaxSessions),
   SumOfSquaredIntraNetworkDelays (MaxSessions),
   MinPreTransmissionDelay (MaxSessions),
   MaxPreTransmissionDelay (MaxSessions),
   SumOfPreTransmissionDelays (MaxSessions),
   SumOfSquaredPreTransmissionDelay (MaxSessions),
   MinWindowToNetDelay (MaxSessions),
   MaxWindowToNetDelay (MaxSessions),
```

```
   SumOfWindowToNetDelays (MaxSessions),
   SumOfSquaredWindowToNetDelays (MaxSessions),
   MinDelay (MaxSessions, MaxLinks),
   MaxDelay (MaxSessions, MaxLinks),
   SumOfDelays (MaxSessions, MaxLinks),
   SumOfSquaredDelays (MaxSessions, MaxLinks)
integer Session
real MeasuredGenerationRate (MaxSessions),
   MeasuredWindowingRate (MaxSessions),
   MeanEndToEndDelay (MaxSessions),
   MeanSquaredEndToEndDelay (MaxSessions),
   MeanIntraWindowDelay (MaxSessions),
   MeanSquaredIntraWindowDelay (MaxSessions),
   MeanPreWindowDelay (MaxSessions),
   MeanSquaredPreWindowDelay (MaxSessions),
   MeanIntraNetworkDelay (MaxSessions),
   MeanSquaredIntraNetworkDelay (MaxSessions),
   MeanPreTransmissionDelay (MaxSessions),
   MeanSquaredPreTransmissionDelay (MaxSessions),
   MeanWindowToNetDelay (MaxSessions),
   MeanSquaredWindowToNetDelay (MaxSessions),
   MeasuredTransmissionRate (MaxSessions, MaxLinks),
   MeanDelay (MaxSessions, MaxLinks),
   MeanSquaredDelay (MaxSessions, MaxLinks)
dimension GenerationRate (MaxSessions)

do while (.true.)
   print *, 'Enter generation rates : '
   read *, (GenerationRate (Session),
      Session = 1, NumberOfSessions)
   TotalGenerationRate = 0.0
   do Session = 1, NumberOfSessions
      TotalGenerationRate = TotalGenerationRate +
         GenerationRate (Session)
   end do ! Total generation rate
   DistributionSum = 0.0
   SessionDistribution (NumberOfSessions) = 1.0
   do Session = 1, NumberOfSessions - 1
      DistributionSum = DistributionSum +
         GenerationRate (Session) / TotalGenerationRate
      SessionDistribution (Session) = DistributionSum
   end do ! Distribution function for session rates

   do while (.true.)
      print *, 'Enter windows : '
      read *, (Window (Session), Session = 1, NumberOfSessions)

      do while (.true.)
         print *, 'Enter Seed : ' ! Seed used : 314159.0
         read *, Seed
         InitialTime = 1

         do Session = 1, NumberOfSessions
            NumberInProcess (Session) = 0
```

```
            GenerationTimeBufferEntry (Session) = 0
            NumberGenerated (Session) = 0
            WindowingTimeBufferEntry (Session) = 0
            NumberWindowed (Session) = 0
            MinEndToEndDelay (Session) = 1.0E+9
            MaxEndToEndDelay (Session) = 0.0
            SumOfEndToEndDelays (Session) = 0.0
            SumOfSquaredEndToEndDelays (Session) = 0.0
            MinIntraWindowDelay (Session) = 1.0E+9
            MaxIntraWindowDelay (Session) = 0.0
            SumOfIntraWindowDelays (Session) = 0.0
            SumOfSquaredIntraWindowDelays (Session) = 0.0
            MinPreWindowDelay (Session) = 1.0E+9
            MaxPreWindowDelay (Session) = 0.0
            SumOfPreWindowDelays (Session) = 0.0
            SumOfSquaredPreWindowDelays (Session) = 0.0
            MinIntraNetworkDelay (Session) = 1000000000
            MaxIntraNetworkDelay (Session) = 0
            SumOfIntraNetworkDelays (Session) = 0
            SumOfSquaredIntraNetworkDelays (Session) = 0
            MinPreTransmissionDelay (Session) = 1.0E+9
            MaxPreTransmissionDelay (Session) = 0.0
            SumOfPreTransmissionDelays (Session) = 0.0
            SumOfSquaredPreTransmissionDelay (Session) = 0.0
            MinWindowToNetDelay (Session) = 1.0E+9
            MaxWindowToNetDelay (Session) = 0.0
            SumOfWindowToNetDelays (Session) = 0.0
            SumOfSquaredWindowToNetDelays (Session) = 0.0
            do i = 1, NumberOfLinksForSession (Session)
               Link = Links (Session, i)
               TransmissionTimeBufferEntry (Session, Link) = 0
               NumberTransmitted (Session, Link) = 0
               if (i .ge. 2) then
                  MinDelay (Session, Link) = 1000000000
                  MaxDelay (Session, Link) = 0
                  SumOfDelays (Session, Link) = 0
                  SumOfSquaredDelays (Session, Link) = 0
               end if
            end do
         end do ! Measure block initialization

      do while (.true.)
         print *, 'Enter FinalTime : '
         read *, FinalTime

         call FCFSScheme (InitialTime, FinalTime) ! Simulate

         RealFinalTime = Real (FinalTime)
         do Session = 1, NumberOfSessions
            MeasuredGenerationRate (Session) =
               Real (NumberGenerated (Session)) / RealFinalTime
            MeasuredWindowingRate (Session) =
               Real (NumberWindowed (Session)) / RealFinalTime
            RealNumberProcessed =
```

```
            Real (NumberTransmitted (Session,
               LastLink (Session)))
         MeanEndToEndDelay (Session) =
            SumOfEndToEndDelays (Session) /
               RealNumberProcessed
         MeanSquaredEndToEndDelay (Session) =
            SumOfSquaredEndToEndDelays (Session) /
               RealNumberProcessed
         MeanIntraWindowDelay (Session) =
            SumOfIntraWindowDelays (Session) /
               RealNumberProcessed
         MeanSquaredIntraWindowDelay (Session) =
            SumOfSquaredIntraWindowDelays (Session) /
               RealNumberProcessed
         MeanPreWindowDelay (Session) =
            SumOfPreWindowDelays (Session) /
               RealNumberProcessed
         MeanSquaredPreWindowDelay (Session) =
            SumOfSquaredPreWindowDelays (Session) /
               RealNumberProcessed
         MeanIntraNetworkDelay (Session) =
            Real (SumOfIntraNetworkDelays (Session)) /
               RealNumberProcessed
         MeanSquaredIntraNetworkDelay (Session) =
            Real (SumOfSquaredIntraNetworkDelays (Session)) /
               RealNumberProcessed
         MeanPreTransmissionDelay (Session) =
            SumOfPreTransmissionDelays (Session) /
               RealNumberProcessed
         MeanSquaredPreTransmissionDelay (Session) =
            SumOfSquaredPreTransmissionDelay (Session) /
               RealNumberProcessed
         MeanWindowToNetDelay (Session) =
            SumOfWindowToNetDelays (Session) /
               RealNumberProcessed
         MeanSquaredWindowToNetDelay (Session) =
            SumOfSquaredWindowToNetDelays (Session) /
               RealNumberProcessed
         do i = 1, NumberOfLinksForSession (Session)
            Link = Links (Session, i)
            MeasuredTransmissionRate (Session, Link) =
               Real (NumberTransmitted (Session, Link)) /
                  RealFinalTime
            if (i .ge. 2) then
               MeanDelay (Session, Link) =
                  Real (SumOfDelays (Session, Link)) /
                     RealNumberProcessed
               MeanSquaredDelay (Session, Link) =
                  Real (SumOfSquaredDelays (Session, Link)) /
                     RealNumberProcessed
            end if
         end do
      end do ! Output-data computation
```

```
do Session = 1, NumberOfSessions
   print *, 'Session = ', Session
   print *, 'MeasuredGenerationRate = ',
      MeasuredGenerationRate (Session)
   print *, 'MeasuredWindowingRate = ',
      MeasuredWindowingRate (Session)
   print *, 'MeanEndToEndDelay = ',
      MeanEndToEndDelay (Session)
   print *, 'MeanSquaredEndToEndDelay = ',
      MeanSquaredEndToEndDelay (Session)
   print *, 'MinEndToEndDelay = ',
      MinEndToEndDelay (Session)
   print *, 'MaxEndToEndDelay = ',
      MaxEndToEndDelay (Session)
   print *, 'MeanIntraWindowDelay = ',
      MeanIntraWindowDelay (Session)
   print *, 'MeanSquaredIntraWindowDelay = ',
      MeanSquaredIntraWindowDelay (Session)
   print *, 'MinIntraWindowDelay = ',
      MinIntraWindowDelay (Session)
   print *, 'MaxIntraWindowDelay = ',
      MaxIntraWindowDelay (Session)
   print *, 'MeanPreWindowDelay = ',
      MeanPreWindowDelay (Session)
   print *, 'MeanSquaredPreWindowDelay = ',
      MeanSquaredPreWindowDelay (Session)
   print *, 'MinPreWindowDelay = ',
      MinPreWindowDelay (Session)
   print *, 'MaxPreWindowDelay = ',
      MaxPreWindowDelay (Session)
   print *, 'MeanIntraNetworkDelay = ',
      MeanIntraNetworkDelay (Session)
   print *, 'MeanSquaredIntraNetworkDelay = ',
      MeanSquaredIntraNetworkDelay (Session)
   print *, 'MinIntraNetworkDelay = ',
      MinIntraNetworkDelay (Session)
   print *, 'MaxIntraNetworkDelay = ',
      MaxIntraNetworkDelay (Session)
   print *, 'MeanPreTransmissionDelay = ',
      MeanPreTransmissionDelay (Session)
   print *, 'MeanSquaredPreTransmissionDelay = ',
      MeanSquaredPreTransmissionDelay (Session)
   print *, 'MinPreTransmissionDelay = ',
      MinPreTransmissionDelay (Session)
   print *, 'MaxPreTransmissionDelay = ',
      MaxPreTransmissionDelay (Session)
   print *, 'MeanWindowToNetDelay = ',
      MeanWindowToNetDelay (Session)
   print *, 'MeanSquaredWindowToNetDelay = ',
      MeanSquaredWindowToNetDelay (Session)
   print *, 'MinWindowToNetDelay = ',
      MinWindowToNetDelay (Session)
   print *, 'MaxWindowToNetDelay = ',
      MaxWindowToNetDelay (Session)
```

```
            do i = 1, NumberOfLinksForSession (Session)
               Link = Links (Session, i)
               print *, 'Link = ', Link
               print *, 'MeasuredTransmissionRate = ',
                  MeasuredTransmissionRate (Session, Link)
               if (i .ge. 2) then
                  print *, 'MeanDelay = ',
                     MeanDelay (Session, Link)
                  print *, 'MeanSquaredDelay = ',
                     MeanSquaredDelay (Session, Link)
                  print *, 'MinDelay = ',
                     MinDelay (Session, Link)
                  print *, 'MaxDelay = ',
                     MaxDelay (Session, Link)
               end if
            end do
         end do ! Output results

         print *, 'Current time = ', FinalTime
         print *, 'Enter "0" to exit run duration loop : '
         read *, Ind
         if (Ind .eq. 0) go to 40
         InitialTime = FinalTime + 1
      end do ! Run duration loop
40          continue
         print *, 'Enter "0" to exit seed loop : '
         read *, Ind
         if (Ind .eq. 0) go to 30
      end do ! Seed loop
30          continue
      print *, 'Enter "0" to exit window loop : '
      read *, Ind
      if (Ind .eq. 0) go to 20
   end do ! Window loop
20       continue
   print *, 'Enter "0" to exit generation rate loop : '
   read *, Ind
   if (Ind .eq. 0) go to 10
end do ! Generation rate loop
10    continue
stop
end
```

```
! The core of simulator FCFS
Subroutine FCFSScheme (InitialTime, FinalTime)

integer FinalTime
integer WrapAroundIncrement, FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
common / globalblock / NumberOfLinks, NumberOfSessions
integer Window
common / simulatorscheme / Window (MaxSessions)
integer TokenBufferLength, OutLinks
common / schemeblock / TokenBufferLength (MaxSessions),
   OutLinks (MaxLinks, MaxSessions)
common / schemearrivals /
   NumberOfPacketsAtSource (MaxSessions),
   NumberOfTokens (MaxSessions)
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
include 'ParameterLB.f77' ! LinkBufferLength
integer QueueHead, QueueTail
common / queueblock / LinkQueue (MaxLinks, LinkBufferLength),
   QueueHead (MaxLinks), QueueTail (MaxLinks),
   NumberOfPackets (MaxLinks)
integer Time, Session, TokenBufferMarker (MaxSessions),
   TokenBuffers (MaxSessions, MaxLinks + 1), Packets,
   SessionServed (MaxLinks), OutLink
real*8 RealTime
save

if (InitialTime .eq. 1) then ! Initialize
   do Session = 1, NumberOfSessions
      TokenBufferMarker (Session) = 1
      do i = 1, TokenBufferLength (Session)
         TokenBuffers (Session, i) = 0
      end do
      NumberOfPacketsAtSource (Session) = 0
      NumberOfTokens (Session) = Window (Session)
   end do

   do Link = 1, NumberOfLinks
      NumberOfPackets (Link) = 0
      QueueHead (Link) = 0
      QueueTail (Link) = 0
   end do ! Initializations complete
end if

do Time = InitialTime, FinalTime
   RealTime = Time - 1
   do Session = 1, NumberOfSessions
      Length = TokenBufferLength (Session)
      Marker = WrapAroundIncrement (TokenBufferMarker(Session),
         Length)
      TokenBufferMarker (Session) = Marker
      if (TokenBuffers (Session, Marker) .eq. 1) then ! Token
         Packets = NumberOfPacketsAtSource (Session)
```

```
            if (Packets .gt. 0) then ! Use token to enter packet
               call Enqueue (FirstLink (Session), Session)
               call Measure (-1, Session, RealTime, 0)
               NumberOfPacketsAtSource (Session) = Packets - 1
            else ! Store token
               NumberOfTokens (Session) =
                  NumberOfTokens (Session) + 1
            end if
            TokenBuffers (Session, Marker) = 0
         end if
      end do ! Window-tokens return to source nodes

      do Link = 1, NumberOfLinks
         call Dequeue (Link, Session)
         SessionServed (Link) = Session
         if (Session .ne. 0)
            call Measure (Link, Session, 0.0, Time - 1)
      end do ! Link loop ends

   ! Generate packets in current slot
      call GenerateArrivals (Time)

      do InLink = 1, NumberOfLinks
         Session = SessionServed (InLink)
         if (Session .ne. 0) then ! Packet arrived
            OutLink = OutLinks (InLink, Session)
            if (OutLink .ne. 0) then ! Session in transit
               call Enqueue (OutLink, Session)
            else ! Session exits, return window-token
               TokenBuffers (Session,
                  TokenBufferMarker (Session)) = 1
            end if
         end if
      end do ! In-link loop ends

   end do ! Time loop ends
   return
   end
```

```
! Generator of Poisson packets in slot for simulator FCFS
Subroutine GenerateArrivals (Time)

integer Time
integer FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
common / globalblock / NumberOfLinks, NumberOfSessions
real*8 Ran, Seed, GenerationTime
common / arrivalblock / Seed, TotalGenerationRate,
   SessionDistribution (MaxSessions)
common / schemearrivals /
   NumberOfPacketsAtSource (MaxSessions),
   NumberOfTokens (MaxSessions)
integer Session, Tokens

RealTime = Real (Time - 1)
TimeGenerated = 0.0
do while (TimeGenerated .lt. 1.0) ! Generate next packet in slot
   TimeGenerated = TimeGenerated -
      Real (log (1.0 - Ran (Seed))) / TotalGenerationRate
   if (TimeGenerated .gt. 1.0) go to 10 ! No more packet in slot
   RandomValue = Real (Ran (Seed))
   do Session = 1, NumberOfSessions ! Generated packet's session
      if (RandomValue .le. SessionDistribution (Session)) then
         GenerationTime = RealTime + TimeGenerated
         call Measure (0, Session, GenerationTime, 0)
         Tokens = NumberOfTokens (Session)
         if (Tokens .gt. 0) then ! Use token to enter packet
            call Enqueue (FirstLink (Session), Session)
            call Measure (-1, Session, GenerationTime, 0)
            NumberOfTokens (Session) = Tokens - 1
         else ! Store packet at source
            NumberOfPacketsAtSource (Session) =
               NumberOfPacketsAtSource (Session) + 1
         end if
         go to 20
      end if
   end do ! Session loop ends
20      continue
end do ! Generation time loop ends
10      continue
return
end
```

```
! Recorder of measurements for simulator FCFS
Subroutine Measure (Link, Session, RealTimeMeasured,
   TimeMeasured)

integer Session, TimeMeasured
real*8 RealTimeMeasured
integer WrapAroundIncrement, FirstLink
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterTB.f77' ! TimeBufferLength
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
integer GenerationTimeBufferEntry, WindowingTimeBufferEntry,
   TransmissionTimeBufferEntry, TransmissionTimes,
   SumOfIntraNetworkDelays, SumOfSquaredIntraNetworkDelays,
   SumOfDelays, SumOfSquaredDelays
real MinEndToEndDelay, MaxEndToEndDelay, MinIntraWindowDelay,
   MaxIntraWindowDelay, MinPreWindowDelay, MaxPreWindowDelay,
   MinPreTransmissionDelay, MaxPreTransmissionDelay,
   MinWindowToNetDelay, MaxWindowToNetDelay
real*8 GenerationTimes, WindowingTimes
common / measureblock / NumberInProcess (MaxSessions),
   GenerationTimeBufferEntry (MaxSessions),
   GenerationTimes (MaxSessions, TimeBufferLength),
   NumberGenerated (MaxSessions),
   WindowingTimeBufferEntry (MaxSessions),
   WindowingTimes (MaxSessions, TimeBufferLength),
   NumberWindowed (MaxSessions),
   TransmissionTimeBufferEntry (MaxSessions, MaxLinks),
   TransmissionTimes (MaxSessions, MaxLinks, TimeBufferLength),
   NumberTransmitted (MaxSessions, MaxLinks),
   MinEndToEndDelay (MaxSessions),
   MaxEndToEndDelay (MaxSessions),
   SumOfEndToEndDelays (MaxSessions),
   SumOfSquaredEndToEndDelays (MaxSessions),
   MinIntraWindowDelay (MaxSessions),
   MaxIntraWindowDelay (MaxSessions),
   SumOfIntraWindowDelays (MaxSessions),
   SumOfSquaredIntraWindowDelays (MaxSessions),
   MinPreWindowDelay (MaxSessions),
   MaxPreWindowDelay (MaxSessions),
   SumOfPreWindowDelays (MaxSessions),
   SumOfSquaredPreWindowDelays (MaxSessions),
   MinIntraNetworkDelay (MaxSessions),
   MaxIntraNetworkDelay (MaxSessions),
   SumOfIntraNetworkDelays (MaxSessions),
   SumOfSquaredIntraNetworkDelays (MaxSessions),
   MinPreTransmissionDelay (MaxSessions),
   MaxPreTransmissionDelay (MaxSessions),
   SumOfPreTransmissionDelays (MaxSessions),
   SumOfSquaredPreTransmissionDelay (MaxSessions),
   MinWindowToNetDelay (MaxSessions),
   MaxWindowToNetDelay (MaxSessions),
   SumOfWindowToNetDelays (MaxSessions),
   SumOfSquaredWindowToNetDelays (MaxSessions),
```

```
   MinDelay (MaxSessions, MaxLinks),
   MaxDelay (MaxSessions, MaxLinks),
   SumOfDelays (MaxSessions, MaxLinks),
   SumOfSquaredDelays (MaxSessions, MaxLinks)
integer Entry, OldTime, Delay
real*8 GenerationTime, WindowingTime
real IntraWindowDelay

if (Link .eq. 0) then ! Packet generated
   InProcess = NumberInProcess (Session)
   if (InProcess .eq. TimeBufferLength) then ! Overflow
      stop 'Time buffer overflow occurred'
   else ! Enter packet generated
      Entry = WrapAroundIncrement (GenerationTimeBufferEntry
         (Session), TimeBufferLength)
      GenerationTimeBufferEntry (Session) = Entry
      GenerationTimes (Session, Entry) = RealTimeMeasured
      NumberGenerated (Session) = NumberGenerated (Session) + 1
      NumberInProcess (Session) = InProcess + 1
   end if
else if (Link .eq. -1) then ! Packet entered window
   Entry = WrapAroundIncrement (WindowingTimeBufferEntry
      (Session), TimeBufferLength)
   WindowingTimeBufferEntry (Session) = Entry
   WindowingTimes (Session, Entry) = RealTimeMeasured
   NumberWindowed (Session) = NumberWindowed (Session) + 1
else if (Link .ne. LastLink (Session)) then ! Enter transmission
   Entry = WrapAroundIncrement (TransmissionTimeBufferEntry
      (Session, Link), TimeBufferLength)
   TransmissionTimeBufferEntry (Session, Link) = Entry
   TransmissionTimes (Session, Link, Entry) = TimeMeasured
   NumberTransmitted (Session, Link) =
      NumberTransmitted (Session, Link) + 1
else ! Packet transmitted on last link in path
   Entry = WrapAroundIncrement (TransmissionTimeBufferEntry
      (Session, Link), TimeBufferLength)
   TransmissionTimeBufferEntry (Session, Link) = Entry
   TransmissionTimes (Session, Link, Entry) = TimeMeasured
   NumberTransmitted (Session, Link) =
      NumberTransmitted (Session, Link) + 1
   GenerationTime = GenerationTimes (Session, Entry)
   EndToEndDelay = TimeMeasured - GenerationTime + 1.0
   MinEndToEndDelay (Session) =
      Min (MinEndToEndDelay (Session), EndToEndDelay)
   MaxEndToEndDelay (Session) =
      Max (MaxEndToEndDelay (Session), EndToEndDelay)
   SumOfEndToEndDelays (Session) =
      SumOfEndToEndDelays (Session) + EndToEndDelay
   SumOfSquaredEndToEndDelays (Session) =
      SumOfSquaredEndToEndDelays (Session) +
      EndToEndDelay * EndToEndDelay
   WindowingTime = WindowingTimes (Session, Entry)
   IntraWindowDelay = TimeMeasured - WindowingTime + 1.0
   MinIntraWindowDelay (Session) =
```

```
   Min (MinIntraWindowDelay (Session), IntraWindowDelay)
MaxIntraWindowDelay (Session) =
   Max (MaxIntraWindowDelay (Session), IntraWindowDelay)
SumOfIntraWindowDelays (Session) =
   SumOfIntraWindowDelays (Session) + IntraWindowDelay
SumOfSquaredIntraWindowDelays (Session) =
   SumOfSquaredIntraWindowDelays (Session) +
   IntraWindowDelay * IntraWindowDelay
PreWindowDelay = EndToEndDelay - IntraWindowDelay
MinPreWindowDelay (Session) =
   Min (MinPreWindowDelay (Session), PreWindowDelay)
MaxPreWindowDelay (Session) =
   Max (MaxPreWindowDelay (Session), PreWindowDelay)
SumOfPreWindowDelays (Session) =
   SumOfPreWindowDelays (Session) + PreWindowDelay
SumOfSquaredPreWindowDelays (Session) =
   SumOfSquaredPreWindowDelays (Session) +
   PreWindowDelay * PreWindowDelay
OldTime = TransmissionTimes (Session, FirstLink (Session),
   Entry)
IntraNetworkDelay = TimeMeasured - OldTime + 1
MinIntraNetworkDelay (Session) =
   Min (MinIntraNetworkDelay (Session), IntraNetworkDelay)
MaxIntraNetworkDelay (Session) =
   Max (MaxIntraNetworkDelay (Session), IntraNetworkDelay)
SumOfIntraNetworkDelays (Session) =
   SumOfIntraNetworkDelays (Session) + IntraNetworkDelay
SumOfSquaredIntraNetworkDelays (Session) =
   SumOfSquaredIntraNetworkDelays (Session) +
   IntraNetworkDelay * IntraNetworkDelay
PreTransmissionDelay =
   EndToEndDelay - Real (IntraNetworkDelay)
MinPreTransmissionDelay (Session) =
   Min (MinPreTransmissionDelay (Session),
   PreTransmissionDelay)
MaxPreTransmissionDelay (Session) =
   Max (MaxPreTransmissionDelay (Session),
   PreTransmissionDelay)
SumOfPreTransmissionDelays (Session) =
   SumOfPreTransmissionDelays (Session) +
   PreTransmissionDelay
SumOfSquaredPreTransmissionDelay (Session) =
   SumOfSquaredPreTransmissionDelay (Session) +
   PreTransmissionDelay * PreTransmissionDelay
WindowToNetDelay = IntraWindowDelay -
   Real (IntraNetworkDelay)
MinWindowToNetDelay (Session) =
   Min (MinWindowToNetDelay (Session),
   WindowToNetDelay)
MaxWindowToNetDelay (Session) =
   Max (MaxWindowToNetDelay (Session),
   WindowToNetDelay)
SumOfWindowToNetDelays (Session) =
   SumOfWindowToNetDelays (Session) +
```

```
      WindowToNetDelay
   SumOfSquaredWindowToNetDelays (Session) =
      SumOfSquaredWindowToNetDelays (Session) +
      WindowToNetDelay * WindowToNetDelay
   do i = 2, NumberOfLinksForSession (Session)
      NewLink = Links (Session, i)
      NewTime = TransmissionTimes (Session, NewLink, Entry)
      Delay = NewTime - OldTime
      MinDelay (Session, NewLink) =
         Min (MinDelay (Session, NewLink), Delay)
      MaxDelay (Session, NewLink) =
         Max (MaxDelay (Session, NewLink), Delay)
      SumOfDelays (Session, NewLink) =
         SumOfDelays (Session, NewLink) + Delay
      SumOfSquaredDelays (Session, NewLink) =
         SumOfSquaredDelays (Session, NewLink) + Delay * Delay
      OldTime = NewTime
   end do ! Link delay loop ends
   NumberInProcess (Session) = NumberInProcess (Session) - 1
end if
return
end
```

```
! Enqueues packet in link-queue in simulator FCFS
Subroutine Enqueue (Link, Session)

integer Session
integer WrapAroundIncrement
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterLB.f77' ! LinkBufferLength
integer QueueHead, QueueTail
common / queueblock / LinkQueue (MaxLinks, LinkBufferLength),
   QueueHead (MaxLinks), QueueTail (MaxLinks),
   NumberOfPackets (MaxLinks)
integer Packets, Tail

Packets = NumberOfPackets (Link)
if (Packets .eq. LinkBufferLength) then ! Overflow
   print *, 'Link ', Link, ' overflow occurred'
   stop
end if
Tail = WrapAroundIncrement (QueueTail (Link), LinkBufferLength)
QueueTail (Link) = Tail
LinkQueue (Link, Tail) = Session
NumberOfPackets (Link) = Packets + 1
return
end


! Dequeues packet, if any, from link-queue in simulator FCFS
Subroutine Dequeue (Link, Session)

integer Session
integer WrapAroundIncrement
include 'ParameterLS.f77' ! MaxLinks, MaxSessions
include 'ParameterLB.f77' ! LinkBufferLength
integer QueueHead, QueueTail
common / queueblock / LinkQueue (MaxLinks, LinkBufferLength),
   QueueHead (MaxLinks), QueueTail (MaxLinks),
   NumberOfPackets (MaxLinks)
integer Packets, Head

Packets = NumberOfPackets (Link)
if (Packets .gt. 0) then ! Link has packets waiting
   Head = WrapAroundIncrement (QueueHead (Link),
      LinkBufferLength)
   QueueHead (Link) = Head
   Session = LinkQueue (Link, Head)
   NumberOfPackets (Link) = Packets - 1
else ! Link idle
   Session = 0
end if
return
end
```

```
! Data used by simulator FCFS for network of Figure 4.1
Block Data Network

include 'ParameterLS.f77' ! MaxLinks, MaxSessions
common / globalblock / NumberOfLinks, NumberOfSessions
integer TokenBufferLength, OutLinks
common / schemeblock / TokenBufferLength (MaxSessions),
   OutLinks (MaxLinks, MaxSessions)
common / functionblock / NumberOfLinksForSession (MaxSessions),
   Links (MaxSessions, MaxLinks)
data NumberOfLinks, NumberOfSessions / 2, 3 /
data OutLinks (1,1) / 2 /, OutLinks (1,2) / 0 /,
   OutLinks (2,1) / 0 /, OutLinks (2,3) / 0 /
data NumberOfLinksForSession (1) / 2 /,
   NumberOfLinksForSession (2) / 1 /,
   NumberOfLinksForSession (3) / 1 /
data Links (1,1) / 1 /, Links (1,2) / 2 /, Links (2,1) / 1 /,
   Links (3,1) / 2 /
data TokenBufferLength (1) / 3 /, TokenBufferLength (2) / 2 /,
   TokenBufferLength (3) / 2 /
end
```

References

[1] Ahuja, V., "Routing and Flow Control in Systems Network Architecture," *IBM Systems Journal,* Vol. 18, No. 2, 1979, pp. 298-314.

[2] Bially, T., B. Gold, and S. Seneff, "A Technique for Adaptive Voice Flow Control in Integrated Packet Networks," *IEEE Transactions on Communications,* Vol. COM-28, No. 3, March 1980, pp. 325-333.

[3] Bially, T., A.J. McLaughlin, and C.J. Weinstein, "Voice Communication in Integrated Digital Voice and Data Networks," *IEEE Transactions on Communications,* Vol. COM-28, No. 9, September 1980, pp. 1478-1490.

[4] Bullington, K., and J. Fraser, "Engineering Aspects of TASI," *Bell System Technical Journal,* Vol. 38, March 1959, pp. 353-364.

[5] Cerf, V.G., and R.E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications,* Vol. COM-22, No. 5, May 1974, pp. 637-648.

[6] Chandy, K.M., and J. Misra, "The Drinking Philosophers Problem," *ACM Transactions on Programming Languages and Systems,* Vol. 6, No. 4, October 1984, pp. 632-646.

[7] Coviello, G.J., and P.A. Vena, "Integration of Circuit/Packet Switching by a SENET (Slotted Envelope Network) Concept," *Proceedings National Telecommunication Conference,* New Orleans, LA, December 1975, pp. 42.12-42.17.

[8] Fuchs, E., and P.E. Jackson, "Estimates of Distributions of Random Variables for Certain Computer Communication Traffic Models," *Communications of the ACM,* Vol. 13, December 1970, pp. 752-757.

[9] Gafni, E.M., "The Integration of Routing and Flow-Control for Voice and Data in a Computer Communication Network," *Laboratory for Information and Decision Systems*

Report 1239, Sc.D. Thesis Dissertation, EECS Department, MIT, Cambridge, MA, August 1982.

[10] Gafni, E.M., and D.P. Bertsekas, "Dynamic Control of Session Input Rates in Communication Networks," *IEEE Transactions on Automatic Control,* Vol. AC-29, No. 11, November 1984, pp. 1009-1016.

[11] Gallager, R.G., P.A. Humblet, and P.M. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Transactions on Programming Languages and Systems,* Vol. 5, No. 1, January 1983, pp. 66-77.

[12] Garey, M.R., and D.S. Johnson, *Computers and Intractability (A Guide to the Theory of NP-Completeness),* W.H. Freeman and Co., New York, 1979.

[13] Gerla, M., and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications,* Vol. COM-28, No. 4, April 1980, pp. 553-574.

[14] Gold, B., "Digital Speech Networks," *Proceedings IEEE,* Vol. 65, November 1977, pp. 1636-1658.

[15] Golestaani, S.J., "A Unified Theory of Flow Control and Routing in Data Communication Networks," *Laboratory for Information and Decision Systems Report 963,* Ph.D. Thesis Dissertation, EECS Department, MIT, Cambridge, MA, January 1980.

[16] Graves, S.C., H.C. Meal, D. Stefek, and A.H. Zeghmi, "Scheduling of Re-entrant Flow Shops," *Working Paper,* Sloan School of Management, MIT, Cambridge, MA, November 23, 1982.

[17] Hahne, E.L., "Round Robin Scheduling for Fair Flow Control in Data Communication Networks," *Ph.D. Thesis Proposal,* Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, April 1985.

[18] Hayden, H.P., "Voice Flow Control in Integrated Packet Networks," *Laboratory for Information and Decision Systems Report 1152,* S.M. Thesis Dissertation, EECS Department, MIT, Cambridge, MA, October 1981.

[19] Hitz, K.L., "Scheduling of Flexible Flow Shops," *Laboratory for Information and Decision Systems Report 879,* MIT, Cambridge, MA, March 1979.

[20] Ibe, O.C., "Flow Control and Routing in an Integrated Voice and Data Communication Network," *Laboratory for Information and Decision Systems Report 1115,* Ph.D. Thesis Dissertation, MIT, Cambridge, MA, August 1981.

[21] Jaffe, J.M., "Bottleneck Flow Control," *IEEE Transactions on Communications,* Vol. COM-29, No. 7, July 1981, pp. 954-962.

[22] Kekre, H.B., C.L. Saxena, and M. Khalid, "Buffer Behaviour For Mixed Arrivals and Single Server With Random Interruptions," *IEEE Transactions on Communications,* Vol. COM-28, No. 1, January 1980, pp. 59-64.

[23] Kogge, P.M., *The Architecture of Pipelined Computers,* Hemisphere Publishing Corporation, McGraw-Hill, New York, 1981.

[24] Mosely, J., "Asynchronous Distributed Flow Control Algorithms," *Laboratory for Information and Decision Systems Report 1415,* Ph.D. Thesis Dissertation, MIT, Cambridge, MA, June 1984.

[25] Orlin, J.B., "The Complexity of Dynamic Languages and Dynamic Optimization Problems," *Proceedings ACM STOC,* Milwaukee, 1981, pp. 218-227.

[26] Oshinsky, D.A., "Use of Fair Rate Assignment Algorithms in Networks with Bursty Sessions," *S.M. Thesis,* Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, May 1984.

[27] Papadimitriou, C.H., and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity,* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[28] Raubold, E., and J. Haenle, "A Method of Deadlock-Free Resource Allocation and Flow Control in Packet Networks," *Proceedings, IEEE 3rd International Conference on Computer Communications,* Toronto, Ontario, Canada, August 1976, pp. 483-487.

[29] Reiser, M., "A Queueing Network Analysis of Computer Communication Networks With Window Flow Control," *IEEE Transactions on Communications,* Vol. COM-27, No. 8, August 1979, pp. 1199-1209.

[30] Rinde, J., and A. Caisse, "Passive Flow Control Techniques For Distributed Networks," *Proceedings, International Symposium on Flow Control in Computer Networks, Versailles, France, February 1979, J.L. Grange, and M. Gien, eds.,* North Holland, Amsterdam, pp. 155-160.

[31] Schwartz, M., and T.E. Stern, "Routing Techniques Used in Computer Communication Networks," *IEEE Transactions on Communications,* Vol. COM-28, No. 4, April 1980, pp. 539-552.

[32] Weinstein, C.J., and J.W. Forgie, "Experience With Speech Communication in Packet Networks," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-1, No. 6, December 1983, pp. 963-980.

[33] Weinstein, C.J., and E.M. Hofstetter, "The Tradeoff Between Delay and TASI Advantage in a Packetized Speech Multiplexer," *IEEE Transactions on Communications,* Vol. COM-27, November 1979, pp. 1716-1720.

[34] Williams, G.F., and A. Leon-Garcia, "Performance Analysis of Integrated Voice and Data Hybrid-Switched Links," *IEEE Transactions on Communications,* Vol. COM-32, No. 6, June 1984, pp. 695-706.